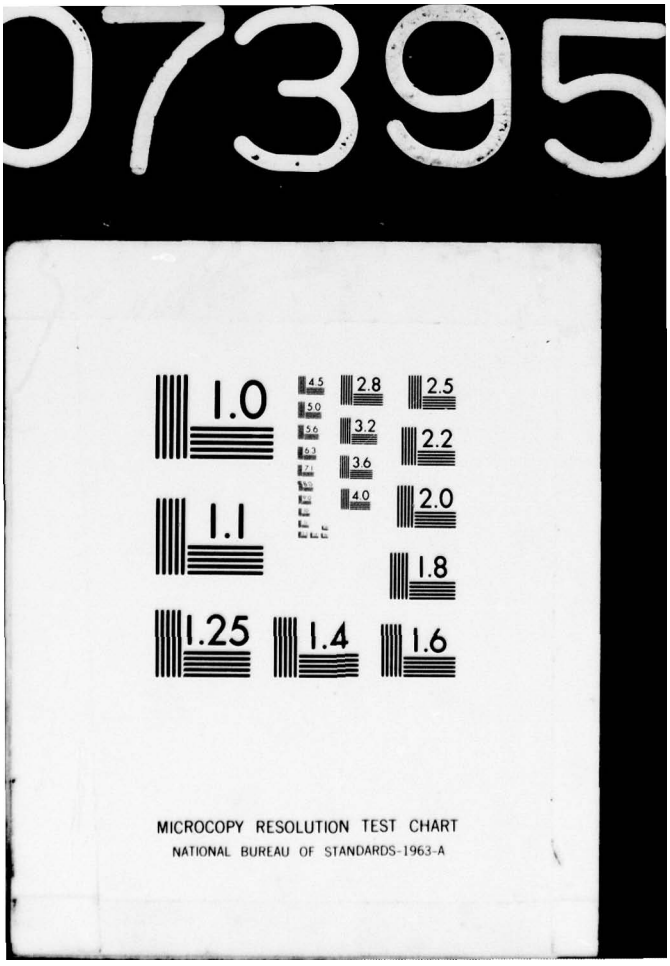
AD-A073 958 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 15.(U)
JUL 79 M L DERTOUZOS N00014-75-C-0661
UNCLASSIFIED LCS-PR-15 NL

1 OF 2
AD A073958


07395
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

LEVEL II

Progress Report 15

July 1977 - June 1978

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: LCS Progress Report 15
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): (6) Laboratory for Computer Science (formerly Project MAC) Progress Report 15 • July 1977 - June 1978

5. TYPE OF REPORT & PERIOD COVERED: DARPA-DOD Progress Report 7/77-6/78

6. PERFORMING ORG. REPORT NUMBER: (14) LCS/PR-15

7. AUTHOR(s): Laboratory for Computer Science Participants; (10) M.L. Dertouzos, Director

8. CONTRACT OR GRANT NUMBER(s): (15) N00014-75-C-0661

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Laboratory for Computer Science
(formerly Project MAC)
Massachusetts Institute of Technology
545 Technology Square, Cambridge, MA 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS:
Advanced Research Projects Agency
Department of Defense
1400 Wilson Blvd., Arlington, VA 22209
(11) 10

12. REPORT DATE: July [illegible] 1979

13. NUMBER OF PAGES: 195

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
Office of Naval Research
Department of the Navy
Information Systems Program
Arlington, VA 22217
(12) 198 p.

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report):
Approved for public release; distribution unlimited
(9) Annual summary rept.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
Jul 77 - June 78,

18. SUPPLEMENTARY NOTES:
Geographically Distributed Systems
Local Network
Single-user Computer
Distributed Operating Systems
Semantics of Distributed Systems
Planning Systems
Data Intensive Planning

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):

Real-time computers	Computer Languages	Automata Theory
On-line Computers	Computer Networks	Morse-Code
Multi-access Computers	Information Systems	Knowledge-Based Systems
Dynamic Modeling	Programming Languages	Complexity
Computer Systems	Computation Structures	Personal Computers

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

Annual summary report of progress made at the Laboratory for Computer Science under this contract during the period July 1977 - June 1978.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409648

Work reported herein was carried out within the Laboratory for Computer Science (formerly Project MAC), an M.I.T. interdepartmental laboratory. During 1977-1978 the principal financial support (65%) of the Laboratory has come from the Defense Advanced Research Projects Agency (DARPA), under Office of Naval Research Contract N00014-75-C-0661. DARPA has been instrumental in supporting most of our research during the last 15 years and is gratefully acknowledged here.

LABORATORY FOR COMPUTER SCIENCE
PROGRESS REPORT 15

JULY 1977 - JUNE 1978

LABORATORY FOR COMPUTER SCIENCE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS 02139

TABLE OF CONTENTS

INTRODUCTION	1
COMPUTER SYSTEMS RESEARCH GROUP	5
A. Introduction	7
B. Distributed Update Management	7
C. Systems Issues in Communications	7
D. Issues in Object-Oriented System	8
DATA BASE SYSTEMS GROUP	13
A. Introduction	15
B. Automatic Data Base Design	15
C. Query Optimization	18
D. Transaction Cost Estimation	21
E. Automatic Data Error Detection	23
F. Automatic Data Error Correction	24
G. Data Base Modeling	26
H. Office Automation	29
DISTRIBUTED SYSTEM SEMANTICS WORKING GROUP	37
A. Introduction	39
B. Study of Applications	41
C. The Target of the Project	43
D. Entities	46
E. Reliability Issues	50
F. Language Constructs for Sending and Receiving Messages	54
G. Protection Issues	58
DOMAIN SPECIFIC SYSTEMS RESEARCH GROUP	67
A. Introduction	69
B. CONSORT: Compile-Time Technology	69
C. MuNet: Object-Time Technology	70
KNOWLEDGE BASED-SYSTEMS GROUP	75
A. Research Summary	77
B. Knowledge Representation and Natural Language Processing	77
C. Natural Language Query to an On-Line Data Dictionary	78
D. Automatic Programming	78
E. Scope of Our Current Work	78
F. Very High Level Language Design and Implementation	79
G. Data Processing System Design	80
H. Automatic Code Generation	81

LOCAL AREA NETWORK WORKING GROUP 85

A. Introduction 87
B. Hardware 87
C. Software 89

PROGRAMMING METHODOLOGY GROUP 93

A. Introduction 95
B. CLU Definitions 95
C. CLU Implementation 96
D. Specification and Verification of Data Abstractions 101
E. Incorporating Abstract Data Types in Stack-Based Languages 101
F. Synthesis of Synchronization Code 105

PROGRAMMING TECHNOLOGY GROUP 121

A. Introduction 123
B. Morse-Code 123
C. Interpersonal Communication 131
D. Other Projects 134

TECHNICAL SERVICES GROUP 145

LABORATORY FOR COMPUTER SCIENCE PUBLICATIONS 149

ADMINISTRATION

Academic Staff

M. L. Dertouzos	Director
J. Moses	Associate Director

Administrative Staff

M. E. Baker	Administrative Assistant
P. G. Heinmiller	Librarian
H. S. Hughes	Administrative Services
E. I. Kampits	Administrative Officer
C. P. Kent	Assistant Fiscal Officer
T. L. Lightburn	Fiscal Officer
G. W. Oro	Fiscal Consultant
G. L. Wallace	Purchasing Agent

Support Staff

G. W. Brown	D. Kontrimus
L. S. Cavallaro	E. Profiro
M. J. Cummings	T. Ramos
S. Geitz	T. Sealy
J. Jones	P. Vancini

INTRODUCTION

This annual report to the Defense Advanced Research Projects agency (DARPA) describes research performed at the M.I.T. Laboratory for Computer Science (formerly Project MAC), funded by that agency and monitored by the Office of Naval Research during the period July 1, 1977--June 30, 1978. The gap between our previous January-based reports and this July-based report is bridged by Interim Progress Report 14/15 which covers the period January 1, 1977--June 30, 1977. Starting with this Progress Report 15, as mutually agreed, we will describe research activities through consecutive annual progress reports coincident with the M.I.T. fiscal year and LCS annual report cycles.

The Laboratory for Computer Science is an M.I.T. interdepartmental laboratory whose principal goal is research in computer science and engineering. Founded in 1963 as Project MAC (for Multiple Access Computer and Machine Aided Cognition), the Laboratory developed the Compatible Time-Sharing System (CTSS), one of the first time-shared systems in the world, and Multics--an improved time-shared system that introduced several new concepts. These two major developments stimulated research activities in the application of on-line computing to such diverse disciplines as engineering, architecture, mathematics, biology, medicine, library science, and management. Since that time, the Laboratory's objectives expanded, leading to research across a broad front of activities that now span four principal areas:

The first such area involves the study and synthesis of intelligent programs by capturing, representing, and using knowledge which is specific to the problem domain. DARPA funded research in this area includes the use of knowledge in programs that comprehend typed natural-language (English) queries and the use of Morse-code knowledge by programs that can detect Morse-code signals in extremely noisy environments.

The second research area has as its purpose the achievement of sizable improvements in the ease of utilization and cost effectiveness of machines, programming languages and systems. It is this research that is predominantly supported by DARPA. In this area the Programming Methodology research group strives to achieve this broad goal through a top-down approach for the development of programs subject to certain constraints that are imposed upon the programmer. Toward the same goal, the Domain Specific Systems research group is exploring the programming of real-time systems from higher-level, domain-specific languages for the control of physical processes. Other research in this area includes the study of very large data bases, the architecture of individual "personal" machines, and the organization of geographically distributed systems of computers. The latter research program is carried out by the Computer Systems and Programming Methodology research groups from the points of view of achieving cohesive applications on interconnected autonomous systems, exploiting the decreasing costs of processors and memories, improving overall performance and reliability, protecting information, and ensuring privacy.

The Laboratory's third principal area of research involves exploration and development of theoretical foundations in computer science and is sponsored primarily by the National Science Foundation.

The fourth area of Laboratory research is entitled Computers and People and entails societal as well as technical aspects of the interrelationships between people and machines. This area is sponsored primarily by industrial organizations.

During the past year, the Laboratory consisted of 221 members--36 faculty, 11 visitors, 56 professional and support staff, 85 graduate and 33 undergraduate students--organized into 14 research groups. The academic affiliation of most of the faculty and students is with the Department of Electrical Engineering and Computer Science. Other departments represented in the Laboratory membership are Mathematics, Architecture, Humanities, The Sloan School of Management, and the Division for Study and Research in Education.

Technical results were disseminated through the publications of the Laboratory members, LCS Technical Reports (TR183-TR201), LCS Technical Memoranda (TM87-TM105), as well as through articles in the technical literature.

Since 1977, geographically distributed systems have evolved into a major Laboratory focus, involving about half of our Laboratory presonnel. Research in this area strives to make possible geographically distributed systems consisting of a large number of processors. The central theme of our research involves local autonomy of each processor, as well as application cohesiveness of the overall system. The theme is pursued at the various levels of representation that characterize this research. In particular, at the hardware level, the Domain Specific Systems research group is developing a single-user computer that will be manufactured for us by the Heath Company; while the Technical Services group is pursuing the network that will link at least 100 of these machines within our Laboratory. At the operating system level, the Domain Specific Systems research group is pursuing research in and development of a distributed operating system that will reside on those machines. The Computer Systems research and Programming Methodology groups are pursuing a general-purpose language especially suited to the semantics of distributed systems. At the applications level our Programming Technology group is researching the structure of a system that makes possible planning in the presence of large amounts of data in the context of energy policy planning.

Michael L. Dertouzos
Director

Assembly and compilation of this DARPA report was done by Paulyn G. Heinmiller under the overall responsibility of Dr. Eva I. Kampits

COMPUTER SYSTEMS RESEARCH

Academic Staff

D. D. Clark,
Acting Group Leader
F. J. Corbato
J. H. Saltzer,
Group Leader*
L. Svobodova

Research Staff

K. T. Pogran
D. Wells

Undergraduate Students

R. Baldwin
H. Carter
N. Chiappa
C. Davis
C. Hornig
J. Maloney
T. McMahon
K. Nyberg
R. Planalp
S. Ratliff
C. Schieck
A. Urbina

Graduate Students

A. Benjamin
E. Ciccarelli
S. Kent
A. Luniewski
A. Mason
A. Mendelsohn
W. Montgomery
D. Reed
K. Sollins

Support Staff

V. Newcomb
M. Webber

Visitors

D. Morgan
A. Takagi

* on leave September 1977 - August 1978

PAGES 3 + 4 BLANK

COMPUTER SYSTEMS RESEARCH

A. INTRODUCTION

During this year, the Computer Systems Research Group was engaged in a variety of projects related to the development of a distributed computing system. The two most important of these projects, the development of a high-speed local network and a preliminary study of the semantics of distributed computing, were performed jointly with other research groups in the Laboratory, and are described under the Distributed System Semantics and Local Area Network working groups. This section describes a number of smaller activities, generally related to distributed systems.

B. DISTRIBUTED UPDATE MANAGEMENT

Work was largely completed on two Ph.D. theses on this area. D. Reed proposed an algorithm for coordinating the update of information items at several physical sites, by creating a single coordinator for any given update. The approach explicitly takes into account the various failures that may occur, and also makes it possible to create new collections of items to be updated in a coordinated manner without changing previous users of those items. W. Montgomery has developed an alternate coordination scheme in which messages requesting updates are properly ordered at several sites by defining a logical communication net through which the messages are sequenced. Finally, A. Takagi, a visiting scientist from Nippon Telepraph and Telephone Corp. (Tokyo, Japan), developed a coordination scheme in which transactions are allowed to use new, yet uncommitted values produced by other transactions. If a transaction aborts, all transactions that used values produced by such an aborted transaction have to be backed out. Mechanisms were developed to handle the back out problem cleanly and efficiently. This scheme increases the effective degree of concurrency in accessing the database while it presents consistency constraints as dictated by a particular application.

C. SYSTEMS ISSUES IN COMMUNICATIONS

S. Kent worked on problems of computer and communication security as part of our distributed systems research effort. One aspect of this work involves the determination of security requirements associated with a class of broadcast communication scenarios which are expected to be employed in distributed systems, and the development and analysis of protocols necessary to achieve these requirements. Another aspect of the work involves development and evaluation of mechanisms to support protected subsystems in "hostile" distributed system sites.

E. Ciccarelli completed his research on the design of network control programs (NCPs). The thesis presenting this research discusses the design of NCPs for operating systems structured around a "security kernel." The design seeks to minimize and simplify the kernel-resident parts of the NCP, so that the dependence of the operating systems's security on the operation of the NCP is reduced and better understood. The thesis presents a general model for an NCP and analyzes sources of network dependence, complexity, and potential security problems. An implementation design for the kernel-resident part of the NCP is presented, primarily network-independent and structured by P. Janson's type-extension discipline. Implementation of the user-domain

parts of the NCP is discussed, demonstrating the network-independence of the kernel, and considering problems of efficiency. In particular, for systems where the general user process cannot provide adequately fast response to incoming messages, two techniques are available: one uses separate, "streamlined" processes to handle frequent simple responses; the other involves a special-purpose "buffer processor" network host, designed to provide the quick response. A very simple protocol for such buffer processors is presented, which interfaces to end-to-end reliable or secure communication protocols in a modular fashion, and which allows buffer processors to remain insecure.

D. ISSUES IN OBJECT-ORIENTED SYSTEM

A. Luniewski is developing an abstract architecture for computers suited for supporting an "object-oriented" language such as CLU. The architecture supports the efficient use of small objects, and permits the uniform use of base-level and user-defined object types. In addition to considering the issue of data abstractions, "objects," the architecture addresses the issues of flow control and control abstractions and attempts to provide a uniform mechanism for the implementation of control abstractions.

K. Sollins is studying certain specific issues that arise in a distributed system supporting object-oriented addressing, in particular the problem of copying objects from one machine to another. When copying an object into a new naming context, it is necessary to insure that names of other objects stored in that object are resolved correctly.

Publications

1. d'Oliveira, Cecilia. A Conjecture About Computer Decentralization. B.S. thesis, M.I.T., Laboratory for Computer Science, LCS/TM-90. Cambridge, Ma., October 1977.

2. Kent, Stephen. "Network Security: A Top Down Approach Shows Problems." Data Communications. June 1978.

3. Kent, Stephen. "Encryption-Based Protection for Interactive User/Computer Communication." IEEE Proceedings 5th Data Communication Symposium. Snowbird, Ut., September 1977.

4. Saltzer, Jerome. "Naming and Binding of Objects." Operating Systems. Lecture Notes in Computer Science, Vol. 60. Edited by R. Bayer. New York: Springer-Verlag, 1978.

5. Svobodova, Liba. "Performance Problems in Distributed Systems." Conference of the Canadian Information Processing Society. Edmonton, Alberta. May 1978.

6. Svobodova, Liba. "Performance Evaluation in View of Changing System Structures." Performance of Computer Installations. Amsterdam: North-Holland (To be Published).

Theses Completed

1. Bradford, Richard. "Linking a Datatrol Credit Management System to an IBM S/370." unpublished B.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, January 1978.

2. Ciccarelli, Eugene. "Multiplexed Communication for Secure Operating Systems." unpublished M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, January 1978.

3. Kauffman, James. "A Design of a One-Pass Interactive Text Formatter." unpublished B.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

4. Krizan, Brock. "A Minicomputer Network Simulation System." unpublished M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, September 1977.

5. Levine, Paul. Facilitating Interprocess Communication in a Heterogeneous Network Environment." unpublished M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, July 1977.

6. McMaster, James. "A Profile System for the Data General Nova." unpublished B.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, July 1977.

7. Selinger, Robert. "Operating System Support for a Data Base Management System." unpublished B.S. thesis. M.I.T., Department of Electrical Engineering and Computer Science. May 1978.

Theses in Progress

1. Montgomery, Warren. "Robust Synchronization of Access to Shared Information in a Distributed System." Ph.D. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, January 1979.

2. Nevins, Russell. "An Efficient Logic Simulator for the Trident Guidance Computer." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, September 1978.

3. Reed, David. "Naming and Synchronization in a Decentralized Computer System." Ph.D. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, September 1978.

4. Sollins, Karen. "Copying in a Distributed System." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, January 1979.

5. Strazdas, Richard. "A Network Traffic Generator for DECNET." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1978.

6. Woltman, George. "Controlling Terminals with High-Level Protocols." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1978.

7. Wyleczuk, Rosanne. "Timestamps and Capability-Based Protection in a Distributed Data Base System." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected data of completion, January 1979.

Talks and Presentations

1. Clark, David. "The Multics Kernel Design Project." ACM Sixth Symposium on Operating Systems Principles. Purdue University, Lafayette, In., November 1977.

2. Clark, David. Session Chairman "Distributed Data Base Implementation." 16th Annual Lake Arrowhead Workshop, Lake Arrowhead, Ca., August 1977.

3. Kent, Stephen. "Encryption-Based Protection for Interactive User/Computer Communication." IEEE 5th Data Communications Symposium, Snowbird, Ut., September 1977.

4. Kent, Stephen. Panelist "Requirements, Theory, and Problems of Network Security." National Telecommunications Conference 77, Los Angeles, Ca., December 1977.

5. Kent, Stephen. "Network and Communication Security." Invited Lecturer North Carolina State University, Raleigh, NC., April 1978.

6. Montgomery, Warren. "Measurements of Sharing in Multics." ACM Sixth Symposium on Operating Systems Principles. Purdue University, Lafayette, In., November 1977.

7. Reed, David. "Synchronization with Eventcounts and Sequencers." ACM Sixth Symposium on Operating Systems Principles. Purdue University, Lafayette, In., November 1977. (To be published in Communications of the ACM.)

8. Reed, David. "Naming and Synchronization in a Distributed Computer System." Xerox Palo Alto Research Center, Palo Alto, Ca., February 1978; University of Southern California, Los Angeles, Ca., February 1978; IBM San Jose Research Center, San Jose, Ca., February 1978; IBM Thomas J. Watson Research Center, Yorktown Heights, NY., March 1978.

9. Reed, David. "Naming of Objects in Distributed Autonomous Computer Systems." University of Minnesota, Minneapolis, Mn., January 1978.

10. Saltzer, Jerome. Panelist "The Role of Performance Modeling in System Design." ACM Sixth Symposium on Operating Systems Principles, Purdue University, Lafayette, In., November 1977.

11. Saltzer, Jerome. Lecturer "Naming and Binding of Objects." Technical University of Munich, Germany, July 28 to August 5, 1977; Technical University of Munich, Germany, March 30 to April 6, 1978.

12. Svobodova, Liba. Panel Session Chairman "Performance Evaluation in View of Changing System Structures." International Conference on Performance of Computer Installations, Gardone Riviera, Lake Garda, Italy, June 1978.

13. Svobodova, Liba. Chairman "Computer Performance Evaluation Applications: Analysis of Distributed Systems." ACM SIGMETRICS/CMG VIII Conference, Washington, D.C., December 1977.

14. Svobodova, Liba. Lecturer, Summer School on Computer Systems Performance Evaluation, Sogesta, Italy, 1978.

Committee Memberships

Clark, David. DARPA IPTO TCP Working Group

Reed, David. DARPA IPTO TCP Working Group

Saltzer, Jerome. DARPA IPTO Working Group

DATA BASE SYSTEMS

Academic Staff

M. Hammer, Group Leader

Graduate Students

E. Cardoza
A. Chan
S. Danberg
J. Kunin
D. McLeod
B. Niamir
S. Sarin
S. Zdonik

Undergraduate Students

B. Berkowitz
J. Dellaquilla
S. Karkula
J. Koschella
R. Leong
H. Shao
D. Slutz
M. Tuceryan
L. Wang
G. Woltman

Support Staff

M. Nieuwkerk

DATA BASE SYSTEMS

A. INTRODUCTION

Our research efforts this year had two principal themes: data base performance and data base semantics. The former area includes such issues as automatic data base design, query optimization, and transaction cost estimation; the latter encompasses our efforts in the automatic detection and correction of data errors, user interface design, and data base modeling. In addition, we initiated this year a project in the area of office automation; we believe this to be a natural outgrowth of our other activities, since most office applications are data intensive.

B. AUTOMATIC DATA BASE DESIGN

Our work in automatic data base design has had three foci: testing and extending earlier work in attribute partitioning; automating the physical design process by utilizing a conceptual schema of the data base; and developing an approach to distributed data base design.

B. Niamir has completed his work on the problem of attribute partitioning in a self-adaptive relational database system. The objective of attribute partitioning is to minimize the volume of information transferred between a random access secondary storage device and primary memory. Partitioning the attributes of a file means storing a subset of the attributes (columns) of a file (relation) together, separate from other subsets of attributes. In a partitioned file environment, when a query requests a group of attributes that have been stored together in the same subfile, only that subfile need be accessed. Attribute partitioning is a viable database design optimization strategy because of the following two reasons:

1 The great majority of queries made to a database request only a subset of the attributes of a file;

2. Most queries made to a database require that more than one tuple be retrieved from the file. Attribute partitioning may be defined as the task of assigning the attributes of a file to the same subfile whenever they are consistently retrieved together. The consequence of attribute partitioning is the localization in the same physical area of information that is predominantly requested together.

The approach we have taken to finding a near-optimal attribute partition of a file, in the context of a given query pattern, is a heuristic one. We use a stepwise minimization state-space heuristic search strategy to determine a locally optimal attribute partition. We have identified a group of such heuristics that have consistently found the optimal partition for example problems (where the optimal partition was known to us).

We have devoted considerable effort to experimentally verifying the desirability of our attribute partitioning heuristics. We have conducted a series of more than 100 experiments. In each experiment we specified the following file and query pattern parameters: the number of attributes in the file (we have considered files with 5, 6, 7, 8, 15, 22, and 30 attributes), the length and selectivity of each attribute, the set of

attributes that are indexed by a secondary index, the number of queries in the query pattern, the frequency of each query, the predicate of each query, and the attributes to be selected and projected by each query. The space of all possible file parameters and query pattern parameters is much larger than what can be fully covered. However, we did range parameter values over a wide spectrum; and the results obtained from our series of experiments have been sufficiently consistent to make us believe that the same results will hold for almost any reasonable specification of the above parameters.

The conclusion we have reached as a result of this program of experimentation may be summarized as follows. For experiments with files of 8 attributes or less (in which we knew the optimal partition of the file by running an exhaustive enumeration procedure that evaluated all possible partitions), the two main heuristics we developed (the fast pairwise grouping heuristic and the single attribute degrouping-regrouping heuristic), when used in conjunction with one another, always found the optimal partition. For experiments with files of more than 8 attributes, it was observed that the above two heuristics found a partition which was significantly superior to the unpartitioned file, and which was superior to partitions found by other heuristic search techniques. This result, and the rapid convergence of the two heuristics in finding the optimal partition, have led us to believe that these two heuristics will consistently find at least a near optimal partition.

The improvement in database performance as a result of attribute partitioning can be significant. The number of page accesses made to an optimally partitioned file is between 40% and 70% of the number of page accesses made when the file is left unpartitioned. For files with a large number of attributes, the improvement in performance is even greater.

Our two attribute partitioning heuristics also operate in modest time, on the order of the number of attributes in the file.

We may also conclude, from the rapid convergence of our heuristics, that most of the advantage of attribute partitioning may be realized by degrouping a few of the most active attributes of the unpartitioned file and storing it separately in its own subfile. If searching for the optimal partition is not computationally practical, then it should be possible to separate the most active attributes of the file and still realize a significant reduction in the database performance cost.

L. Wang has been extending previous results on attribute partitioning and index selection. In our earlier work on attribute partitioning, it was assumed that the set of attributes that were indexed would remain constant during the course of searching for the optimal partition; in our index selection studies, it was assumed that all attributes of the files were stored together in a single file. By simultaneously considering the problems of finding the optimal attribute partitioning and finding the optimal index set for a file, it is hoped that the resulting partition (and its accompanying set of indices) will have a still lower performance cost, compared to when the attribute partitioning and the index selection are considered as two independent problems. A new heuristic based on the attribute partitioning heuristic has been developed and will be experimentally verified.

A. Chan has been working on the problem of optimizing the performance of integrated databases. An integrated database is a collection of data used for a variety of application functions in an enterprise. To cope with the evolution of applications, the physical organization of an integrated database must be optimized for the prevailing access requirements. By providing a non-procedural interface between the application programs and the database, it is possible to tune the performance of the database, by adjusting its physical representation, and still have only minimal impact on the logic of the application programs. Among the conventional (hierarchical, network and relational) data models, the relational approach comes closest to providing users with a logical view that is independent of the actual storage structures used to represent the data, and that facilitates the reorganization of data bases and reoptimization of queries against them. However, the implementations of relations in current relational systems have not been supportive of performance levels competitive with databases based on the network or hierarchical data models. We believe this is partly due to the failure of current relational systems to use more complex storage structures, such as partitioning a relation vertically or horizontally in its physical representation, or using physical pointers to provide rapid access paths between relations. A more important reason may be the failure to introduce certain kinds of redundancy at the physical level in order to eliminate or reduce the cost of cross-referencing between relations. For example, it may be desirable to replace the representation of two third normal form relations by their join at the physical level. However, to do so requires the semantic knowledge that the two relations are non-loss joinable.

It is our objective to investigate a wide range of implementation alternatives for relational databases and to automate the physical design process for them. Because of the vast design space that modern file organization techniques provide, the use of formal mathematical programming techniques to search for the optimal organization is not a feasible approach. Instead, we are developing heuristics for the synthesis of a good, stable physical organization for the prevailing access requirements. Our approach is iterative and goal directed. From the knowledge of the semantic concepts represented by the relations, an initial tentative implementation is selected. This is then successively modified and refined by examining the performance bottlenecks it presents in the processing of the transactions that constitute the data base usage pattern. The automatic designer will be provided with knowledge regarding the types of perturbations of a physical organization that may relieve a given type of processing bottleneck.

We feel that a logical schema described purely in relational terms does not provide enough semantic information for this design process, information which is usually available to, and used by, a human designer. Our approach, therefore, is to provide input to the automatic designer of the conceptual schema of the database, which contains information about the semantics of the database. The designer will then generate (algorithmically) a relational schema for the database; this will be the schema with which application programs will interact. At the same time, semantic information that might be useful to the physical design process will be extracted. For example, this will allow the automatic designer to decide whether derived information as expressed in the conceptual schema should be recomputed when required, or should be represented via controlled physical redundancy.

An important objective for database integration is improved consistency in the stored data. Therefore, we intend to include in the conceptual schema the specification of some basic validity constraints on the database, ones which will be automatically maintained by the database system during database modifications. We believe that certain types of constraints are more fundamental than others (in the sense that they are application independent), and that these should be directly embedded in the data model to simplify the specification of the usage pattern and to guide the physical database design process.

The descriptions of the conceptual schema used by the automatic designer is couched in terms of the Semantic Data Model, which was developed by our group and is described in a subsequent section.

E. Cardoza is studying the data base design problem for distributed relational database management systems. The major aim of this work is to allocate the data base among the different sites of the network so as to minimize such costs as response time, storage charges, and updating costs. Although a great deal of literature has appeared on the file allocation problem for networks, this earlier work has ignored a number of important factors in modern distributed data base management.

1. For example, user accesses have typically been modeled simply as accesses to a single file from a given node; queries involving two different relations (or files) are not directly modeled. Thus the cost advantage of a single site having two different relations which are often used together is not taken into account.

2. The cost effects of synchronization mechanisms for performing updates of files with multiple copies is not modeled in earlier work.

3. Earlier models assume that a complete file is the unit of assignment to the various sites. The possibility of reducing costs by allowing vertical and/or horizontal partitioning of files in the assignment of the data base has not been considered.

We are engaged in the preliminary design of a system to select near-optimal distributions for a distributed data base, taking into account the above factors. A crucial part of this system is an evaluator that assesses the cost of any proposed distribution in the context of a given usage pattern. The focus of our initial effort is on a design system for the SDD-1 distributed data base system.

C. QUERY OPTIMIZATION

We have been engaged in a number of efforts relating to the optimization of queries made against a data base. The first relates to determining which of several processing techniques for a given high-level query is most effective, in the context of a particular data base organization. James Koschella and Edward Cardoza have considered this problem in the context of the Datalanguage data access language of the Datacomputer. The specific problem that has been addressed is the following. For purposes of the study, a query in Datalanguage is considered as having the form

For X1 in R1 with B1(X1)

For X2 in R2 with B2(X1, X2)

. . . . .

For Xn in Rn with Bn(X1, X2, ..., Xn)

where X1, X2, ..., Xn are tuple variables which range over (respectively) the relations R1, R2, ..., Rn of the data base, and where Bj(X1, X2, ..., Xj), for each j is a Boolean predicate on the tuple variables X1, ..., Xj.
The problem is to find a reordering (Xi_1, Xi_2, ..., Xi_n) of X1, ..., Xn such that the query expressed in the form

For Xi_1 in Ri_1 with $Bi_1'(Xi_1)$
For Xi_2 in Ri_2 with $Bi_2'(Xi_2)$

. . . . .

For Xi_n in Ri_n with $Bi_n'(Xi_n)$

would entail the minimal amount of cost (of all reorderings) using the query processing strategy of the Datacomputer.

A number of heuristics, similar in spirit to the Decomposition Method of Wong and Youseffi, have been proposed which find a "good" if not an optimal reordering in an efficient manner. A study and validation of these heuristics is currently being conducted.

In a similar vein, S. Danberg has been developing techniques for concurrently evaluating multiple queries against a single data base, overlapping their processing requirements.

S. Zdonik has been studying the use of data base semantics for the optimization of data base queries. Contemporary data base languages enable users to express queries in terms of predicates that the selected data must satisfy. Conventional query optimization techniques seek only to find the most efficacious way of using available access structures to answer the given query; our approach attempts to exploit the semantics of the query, either in processing it or in transforming it into an alternative form, semantically but not syntactically equivalent to the original, which can be processed in a way more efficient than any means of answering the original. Obviously, such transformations must preserve the meaning of the query and they must take into account the available access structures that can be used in answering queries.

Consider the following examples.

1. Get the names of all employees whose job type is pilot.
   If all pilots work in the flight department, then the query could become:
   Get the names of all employees of the flight department whose job type is pilot.
   This is a desirable transformation if determining the members of the flight department is a relatively inexpensive operation, and if the number of employees in the flight department is much less than the total number of employees.

2. Get all the ships whose maximum speed is greater than 25 knots. Suppose that a ship is either a tanker, a merchantman, or a naval ship, and that the maximum speed of a tanker is 15 knots and of a merchantman 20 knots. Then the original query can be rephrased as "get all naval ships whose maximum speed is greater than 25 knots." This transformation may improve query processing if there is some file structure (such as an inversion) that makes finding naval ships an efficient operation.

The main features of our approach to semantic query optimization are the following:

1. These techniques are not concerned with processing strategies based on properties of Boolean connectives or operations on data structures. That is the province of conventional optimization techniques.

2. Semantic query optimization will usually produce a new query that is a transformation of the original text. The new text may involve conditions on *different semantic structures than those used in the original query*, but it must be semantically equivalent to the original in the sense of producing the same output set.

3. In semantic query optimization, we will utilize domain specific knowledge to perform transformations. This knowledge is expressed in the form of predicate calculus constraints, statements of conditions that must be met by all legal *configurations of the data base*. These constraint expressions are related to the semantic integrity predicates used for data error detection.

4. In order to make use of some rich semantic information, we employ an effective data model that is higher level than the relational model, but that can easily be supported by a relational system. This model is based on the notions of entities and associations and is closely akin to the Semantic Data Model developed in our group and described in this report.

We have identified several *different kinds of optimizations*. Some of these transformations are:

1. Term Replacement

   Here, the goal is to substitute for one or more terms in the original query a collection of terms that are equivalent to the originals but are less costly to evaluate. To achieve these replacements, one needs to use constraints of the form: P1 <=> P2, where P1 and P2 are predicates on entities in the data base schema.

2. Term Addition

   Here, one adds terms to the original query, with the goal of having these new (and inexpensive) terms evaluated before evaluating existing expensive terms in the query; this should reduce the scope of application of the expensive term. The cost of evaluating these new terms must, of course, be less than the savings

achieved in the processing of the old expensive terms. The kind of constraint that is useful here is of the form: $P_1 \Rightarrow P_2$. In this case, if a query contains P_1 as a subexpression, then adding P_2 to it as a conjunct does not change its meaning.

3. Term Modification

This optimization transforms an existing term in the query into an alternate form, one that involves revised conditions on the same entities. The new form might be more efficiently evaluated or might be amenable to further semantic optimization.

4. Processing Strategies

This class of optimizations does not entail source level transformation, but is based on passing additional information to the search engine that actually retrieves records of interest. For example, if we know an upper bound on the number of records that satisfy a condition, then we can instruct the search engine not to attempt to find more than that number, enabling it to terminate some searches early.

We are engaged in devising an architecture for a system that, given a query, will decide what optimizations might be desirable to apply to it, search for relevant constraints, and effect cost-improving transformations.

D. TRANSACTION COST ESTIMATION

Underlying all of our work in database system performance is the notion of a transaction cost estimator, a system that can predict the cost that a DBMS will incur in the processing of a transaction. We have developed the underlying technology for such a system, and have sought to test its applicability to operational data base systems. To that end, Brian Berkowitz has been working on a transaction cost estimator for the Datacomputer, an operational DBMS that supports a relational data model and allows for the construction and maintenance of large databases. The goal of this work is to produce an estimator that will examine a Datacomputer transaction and produce an estimate of the amount of time needed to process it. The transaction cost estimator will use a statistical description of the contents of the data base as well as parameters describing system load in order to produce an estimate of the amount of time that will elapse between the time when the transaction is issued and the time when the Datacomputer has completed processing the transaction. A transaction cost estimator could be used to provide the user of the Datacomputer with an estimate of how long it would take the Datacomputer to process a proposed transaction; it would also be useful in an automatic database design system.

Our efforts so far have been directed towards estimating the amount of time necessary to process retrieval transactions. These are transactions that examine records in a database but do not add or delete records or change the contents of any records. We have identified three basic components of the processing of a retrieval transaction. These are the number of page accesses needed to read pages containing data utilized by the transaction, the amount of cpu time necessary to process the

transaction, and the page faults that occur when code or data accessed in processing the transaction is swapped out because of high demand for memory in the Datacomputer's multiprogramming environment. These page faults do not count the page accesses necessary to read in records used by the transaction. The first two parameters are dependent only on the translation and the contents of the database. The number of page faults is dependent on the transaction, the contents of the database, and also the total demand for pages by all current users of the Datacomputer. The three parameters can be combined with a statistical description of system load to produce an estimate of the total time needed to process the transaction.

We have developed cost estimators to estimate the number of pages read in retrieving data and also the amount of cpu time required to process a transaction. We have considered two cases in estimating the number of pages read by a transaction. The first case is where the records to be retrieved are randomly distributed throughout the file. Modifying a previously developed page accessing function we have developed a formula which predicts the number of pages read in this case. The formula is derived using combinatorial techniques, and produces its estimate based on the number of records in a file, the number of records on a page, and the number of records to be retrieved.

The second type of file that we have considered is a clustered file. If a file is clustered on a field F, then all records with the same F value are located near each other. If we retrieve records with field F equal to some value k, only a small portion of the file will have to be read. It is also possible for a file to be clustered on several levels. The primary example of this is a multi-level sorted file. If a file is sorted using key MONTH, DAY, and TIME, then the file is clustered on MONTH, clustered on MONTH,DAY (i.e. all records with the same MONTH and DAY values are near each other) and clustered on MONTH,DAY,TIME. We have developed an estimator which predicts the number of pages read in reading records that satisfy a particular predicate from a multi-level clustered file. The estimator converts the predicate into a more convenient form for processing (a disjunction of conjuncts such that no record in the file satisfies more than one conjunct). The converted predicate is then used to produce an estimate using a hierarchical model of the file and utilizing a statistical description of the file's contents.

This estimator was implemented and experiments were conducted to test its validity. A database was constructed and over 800 transactions were considered. The estimated number of pages retrieved was compared to the actual number of pages retrieved in processing each transaction. The average error was small (about 7%)

An estimator was also built to predict the cpu time required to process a transaction. The Datacomputer processes a transaction by first compiling the transaction into machine code, which is then run. The cpu time used to process a transaction is therefore the sum of the cpu time used to compile the transaction and the cpu time used in running the compiled code. An estimate of the cpu time required to compile the transaction is generated by an estimator using some measures of the complexity of the code (e.g., number of lines of code). This estimator was developed using an empirical study of the cpu time required to compile many different types of transactions. The cpu time required to run the compiled code is estimated by using a

"shadow compiler". The estimator we have built simulates the compiler. It parses the transaction, just as the Datacomputer compiler does, and uses the parsed transaction in a "code generator" phase. Where the code generator in the Datacomputer generates a code fragment, the estimator we have built generates estimates of the amount of time necessary to run that particular code fragment. The estimator uses knowledge about how the Datacomputer compiles a transaction, and also statistical descriptions of the contents of files, in order to produce an estimate of the total cpu time required to run the machine code generated for a transaction.

E. AUTOMATIC DATA ERROR DETECTION

We have continued our work in the area of database error-detection; our principal focus here is on building an assertion-based error-detection system. In our approach, the possible error states of a database are described, by the DBA or some other authority, in terms of semantic integrity assertions. The database system will then assume responsibility for detecting any violations of these assertions that are caused by updates to the database. When compared with existing ad-hoc techniques (where "edit-routines" are written by hand for each of the update transactions on the database), this assertion-based approach to integrity checking has the advantages of increased reliability and modifiability.

The main problem with this declarative approach to integrity checking is one of performance. The obvious way to monitor a set of integrity assertions is to reevaluate each assertion every time the database is updated, and this can be prohibitively expensive.

Our approach to the performance problem in assertion-based integrity checking is to "compile" assertion-monitoring procedures based on an analysis of the effects of anticipated update transaction types on the given set of assertions. We assume that the update transactions on a database belong to a limited set of frequently-performed operation types. For each such operation type, a detailed analysis is performed that results in the generation of an efficient procedure that can be used to precisely identify the assertions that may be violated whenever an operation of the given type is invoked by a user of the database. The procedures generated have the following features that contribute to efficient error-detection:

1. They are comparable in efficiency with integrity-checking procedures that an intelligent application programmer might write for the given operations and assertions.

2. When a user attempts to perform an update operation, the associated integrity-checking procedure can be run before the database is modified. Thus, if assertions are violated and it is found necessary to reject the update, there is no need for the database system to perform expensive backing out of the update.

We have been working on the design and implementation of an assertion processing system that performs the above generation of efficient integrity-checking procedures. In our design, this synthesis proceeds in two stages:

1. A logical analysis phase, which generates a set of alternative test procedures for each operation type.

2. An optimization and selection phase, where for each operation type a single test procedure is selected from the associated set of alternatives, based on its expected cost of execution in the context of the underlying physical representation of the database.

The first stage above, referred to as "perturbation analysis," was discussed in a previous Progress Report. S. Sarin has recently completed the design of an algorithm that performs this analysis. The algorithm takes as input a database schema definition, a set of update operations, and a set of integrity assertions, all expressed in high-level terms that are independent of the physical representation of the database. For each update operation, the algorithm generates "perturbation information" for the component expressions of the various assertions, which describes how the values of these expressions are affected by the given database update. This information is then used to construct a set of alternative tests (expressed in the same high-level nonprocedural language as the assertions) for the assertions under the operation. The main thrust in this algorithm is the identification of conditions under which it can be proved that the operation will not violate a given assertion; the conditions identified are such that they can be efficiently tested when the update operation is invoked, and thus lead to efficient tests of the assertions.

This perturbation analysis algorithm has been implemented by D. Slutz, using the language MDL (an extended LISP) on a PDP-10 computer. The procedure was tested on some example database definitions (including descriptions of update operations and integrity assertions), and produced the desired results. Testing of this system will continue.

For the second phase of the assertion processing system, R. Leong has been working on the development of a "test selection" procedure. The input to this procedure consists of the output of the perturbation analysis phase (namely, the sets of alternative tests associated with the update operations), plus a description of how the database is represented in terms of file structures and access methods. For each update operation, the procedure then selects a test that has the lowest (or close to the lowest) expected cost of execution among all the tests in the set of alternatives associated with the operation. The selection procedure translates the high-level descriptions of the tests into database interactions expressed in the data manipulation language of the underlying database management system; it then uses a transaction cost estimator to determine the expected costs of performing these interactions whenever an operation of the given type is invoked. (For the initial design of the test selection procedure, we have assumed that the database is implemented on the Datacomputer. However, we expect that the techniques we have developed will be applicable to other database management systems as well.)

F. AUTOMATIC DATA ERROR CORRECTION

S. Zdonik has been involved in the design and implementation of a front-end processor (called Data Doctor) that does automatic error correction of database transactions. This type of system is particularly useful in an environment in which

large volumes of data are being keyed by hand into a database, and where manual correction of detected errors is impractical.

Our goal is to build a system that can automatically detect and correct many of the errors that occur when a transaction against a data base is keyed into the system. After detecting that an error exists in a transaction, the system will attempt to identify the precise location and nature of the error, and then seek to correct it, based on an analysis of the erroneous value and of the (presumably correct) contents of the data base. (Of course, the feasibility of this step is limited by the amount of redundancy in the transaction and in the data base.) We draw heavily on a semantic model of the data base to detect erroneous data values and to guide the correction process. Our approach to error correction is heavily based on the premise that erroneous data values are caused by a set of "error mechanisms," a (relatively) small and knowable set of events that can cause the transformation of a correct value into an incorrect one. Typical error mechanisms might include character duplication, character transposition, confusion of visually similar characters, and so on. Once the locus of an error has been established, we employ an analysis of the various mechanisms to determine which is most likely to have been the cause of this particular error. Then the effects of that mechanism can be "undone" in order to reconstruct the original uncorrupted value.

Data Doctor detects errors by means of constraints, predicates that express conditions on the data base that must hold after every transaction with it. When a transaction is submitted to Data Doctor, it evaluates these constraints that are relevant to the transaction; the failure of any of these indicates the possibility of an error in the transaction. In our system, each constraint evaluates to a value between 0 and 1, 0 representing certain data error and 1 indicating perfect data correctness; an error condition is signalled by a number of constraint evaluations returning values below a given threshold. In this situation, the system analyzes the "symptoms" of the faulty transaction and attempts to "diagnose" the nature of its error.

The major system modules are summarized in the following list.

1. Constraints Checker

   The constraints checker is responsible for evaluating those constraints relevant to the submitted transaction.

2. Likelihood Evaluator

   The likelihood evaluator decides whether or not the scores returned by the previous module indicate a situation that requires error correction or whether the report represents a real but unlikely situation.

3. Locus Finder

   The locus finder applies certain heuristics to the evolving pattern of constraint failures to establish a list of possibly erroneous fields in the transaction. An example of a heuristic that might be used by this module is that if all fields after field n are involved in constraint violations but none of the fields before and

including field n are, then we should suspect field 2n+1 as the site of a missing field or a missing separator.

4. Mechanism Chooser

The mechanism chooser selects the most likely error mechanism and the most likely place to apply it, in order to produce a new set of transaction values that can be re-entered into the constraints checker.

The above modules are listed in the order in which they are invoked during the operation of the Data Doctor System. The system continues to call these modules until a set of data values that satisfies the Likelihood Evaluator has been found or until the known set of mechanisms has been exhausted. In the latter case, the set of values that produced the best score will be selected as the best guess. In practice, Data Doctor's corrected values would be reviewed by a human authority before being installed.

We have completed an initial implementation of the system that assumes that there is at most one error in a transaction. G. Woltman, M. Tuceryan, and S. Zdonik have also been investigating the design of an error correction system able to handle multiple errors in a given report. We have investigated two special cases of this problem in some detail.

1. Two errors in the same field (i.e., two error mechanisms applied at the same locus).

2. Two errors in semantically related fields

In the first case, the brute-force method of applying all the possible mechanisms to all possible positions was immediately discarded as impractical. Instead, a set of heuristics was developed to eliminate certain combinations of mechanisms based on observations of the suspicious value. Furthermore, a method of using predicated values for the erroneous field was designed.

In the second case, we considered errors in two fields that are related with respect to some set of constraints. It is possible for the two errors to occur in such a way that the constraints that check their consistency do not notice that anything is amiss. Here, a procedure for determining the possible field combinations that could be in error was developed.

G. DATA BASE MODELING

The conventional approaches to the structuring of data provided by contemporary data base management systems are in many ways unsatisfactory for modeling data base application environments. The features they provide are too low-level, computer-oriented, and representational to allow the semantics of a data base to be directly expressed in its structure. D. McLeod has designed the *semantic data model (SDM)* as a natural application modeling mechanism that can capture and express the structure of an application environment. The design of the SDM is specifically based on a detailed analysis of the most important semantic problems of

conventional data base structuring mechanisms.

It is intended that the features of the SDM correspond to the principal intentional structures naturally occurring in contemporary data base applications. The SDM provides a rich but limited vocabulary of data structure types and primitive operations, striking a balance between semantic expressibility and controlled complexity. Furthermore, facilities for expressing derived (conceptually redundant) information are an essential part of the SDM; derived information is as prominent in an SDM data base as is primitive data.

In brief, an *SDM data base* is a collection of classes. A *class* is a collection of *entities*. The structure of an SDM data base is defined by an *SDM schema*, which describes the classes that constitute it. Classes are collections of entities that are meaningful in the application environment. (Examples here are selected from a data base used to support the monitoring of ships with potentially hazardous cargoes entering U.S. coastal waters.) Classes are used to model collections of objects (SHIPS), events (OIL_SPILLS), "type" abstractions (SHIP_TYPES), aggregates of other entities (CONVOYS), and "values" (SHIP_NAMES, HULL_NUMBERS).

Each class has a collection of *attributes* associated with it, whose purpose is to describe the members of that class or the class as a whole. There are three types of attributes:

1. *Member attributes* describe aspects of each member of a class, by linking the member to one or more related entities in the same or another class. For example, the members of class SHIPS have attributes Name, Home_port, and Engines, which give the ship's name, its home port, and link it to its engines (respectively).

2. An attribute of each member of a class that has the same value for all members of that class is a *class-determined attribute*. Such an attribute is a member attribute, but it is associated with the class as a whole because the attribute has the same value for all class members. For example, to capture the fact that no oil tanker can sail faster than some top speed, the class-determined attribute Absolute_top_speed of class OIL_TANKERS would be defined.

3. A *class attribute* describes a property of a class taken as a whole. For example, the class PORTS has the attribute Number_of_ports, which gives the number of ports currently in the class.

Derived, as well as primitive classes are prominent in typical SDM data bases. Not only is the class SHIPS defined in the example data base, but so is OIL_TANKERS (a subclass of SHIPS). Analogously, derived attributes can be defined in terms of primitive ones. For example, one might define the attribute Inspections of OIL_TANKERS as the inversion of the attribute Tanker_inspected of INSPECTIONS. A comprehensive vocabulary of types of class and attribute derivation specifications which are directly useful in supporting the easy definition of derived information in an SDM data base has been developed. The principal

subclass definition primitive is "restrict," and is used to define OIL_TANKERS in the example above. Similarly, "invert" is used to define the derived attribute Inspections of OIL_TANKERS. These are only examples of the full spectrum of definitional primitives provided by the SDM.

The SDM is designed to enhance the effectiveness and usability of data base systems in the following ways:

1. SDM data bases are to a large extent self-documenting, in the sense that the description and structure of a data base is expressed in terms which are close to those used by users in describing the application environment.

2. The SDM can support powerful user interface facilities, and can improve the user interface effectiveness for a variety of types of users (who have varying needs and abilities). Significantly, SDM data bases capture information in a form accessible to its users, and allow derived information helpful in new data base uses to be defined in the data base structure.

3. The SDM can be used as a tool in the data base design process. The SDM aids in the identification of relevant information in a data base application environment, as well as in organizing that information and relating it to its possible uses. This can greatly improve the design of lower-level, conventional data bases.

One focus of our research has been on data base user interface facilities based on the SDM. First, a powerful semantics-based data base query and update language for the SDM (called the *interaction formalism*) has been designed. This language provides a rich but limited set of built-in data base operations, and allows user-defined transactions to be defined in terms of these primitives. The combination rules are simple, but the vocabulary of primitives allows a good deal of flexibility in describing data base retrievals and modifications.

The SDM also supports an incremental, interactive interface for the "naive" nonprogrammer, called the *interaction formulation advisor (IFA)*. The IFA guides a user through the data base in the process of formulating a query or update request against it. The IFA assumes that the user is largely naive of the data base content and structure, and that the user has limited experience with computerized data bases. The operation of the IFA relies heavily on the SDM data base description and structuring primitives, e.g., derived class and derived attribute specifications. In addition, the algorithm used by the IFA embodies a specific structured, stepwise methodology for expressing data base queries. It is by means of this methodology that the IFA can provide effective support of users naive of a data base's detailed content and structure. A prototype IFA is currently running, and is being extended to handle a wider spectrum of user needs.

In addition to this application, the SDM has been used by a number of other projects in our group as a means of describing the semantics of a data base. We anticipate that the need for such a semantic description mechanism will grow as ever more powerful capabilities are demanded of data base systems.

H. OFFICE AUTOMATION

We have begun a research effort involving several significant issues in the application of computer systems to the office domain. The term "office automation" has recently become so pervasive as to lose any precise meaning; we interpret it to mean the application of computer-based systems to enhance the productivity of office personnel and the efficiency of office operations. Current commercial entries in the office automation field include such tools as word processing and electronic mail systems. It is our contention, however, that computer scientists must look beyond products of this sort to gain the understanding of the underlying nature and purpose of offices that is necessary to have the greatest impact on office operation.

The major activities in offices include information processing, communication, record keeping and decision making. General-purpose devices such as calculators and copying machines, as well as the more recent developments mentioned above, support workers in performing a variety of tasks necessary to accomplish these activities. Such tools mechanize tasks; they do not automate function. We feel that major enhancements in office productivity will derive not from task mechanization, but from the study, and subsequent automation, of office functions.

In order to bring computer technology most effectively to bear upon offices, it is necessary to develop an understanding of the nature of office activities. A crucial dimension in the analysis of this domain is task structure. A structured task is one that is amenable to algorithmic specification and description. Unstructured tasks are those which cannot be so described because they inherently require human intelligence and judgment. Most office procedures consist of a mixture of both kinds of functions. For example, the processing of a purchase requisition might first involve some highly structured processing to assure that the requisition is valid. This would be followed again by an unstructured decision by a buyer to determine how and where the goods should be purchased, based upon complex criteria involving past experience and personal contact with vendors, outstanding or anticipated orders, etc. This decision is followed by further structured activities which produce a purchase order, send copies to various offices, and maintain appropriate records.

It is our thesis that the greatest gains in office productivity can be achieved by systems which both automate the structured, routine activities in an office procedure, and provide office workers with an integrated working environment and a powerful set of tools so that activities which cannot be automated can be carried out more effectively. The goal of an office automation system should not be merely to eliminate paper from existing tasks, but rather to implement as effectively as possible the essential purpose and function of the office. We believe that the analysis and specification of office procedures, the design of support tools, and the organization of the office system as a whole, should be based upon this underlying perception of the role of computer systems in the office.

Therefore the goals of our research are twofold: to develop powerful computer-based tools for the integrated electronic office environment, in order to provide support for office workers performing intelligent decision-making functions; and to investigate tools and techniques for the analysis of office function and the

design of computer systems to implement those functions. We also plan to construct a testbed environment in which to test, analyze and assess these developments in a realistic environment.

A major component of our work is the study of a large number of office situations, in order to develop an understanding of office functions, to identify applications where our efforts can provide high impact, and to build up a case file for subsequent development work. J. Kunin has completed an initial set of office case studies, which give detailed procedure descriptions of about fifteen offices at MIT. In addition, S. Karkula has completed an undergraduate thesis describing the administrative operations of a television station.

As we have noted, while the current state-of-the-art in office systems consists of independent tools which mechanize individual tasks, future technology will involve an integrated electronic office environment. We are developing the design for an interface to this system, a work station which we have termed the "electronic desk." The major issues involved in this work are the human engineering of the interface, and the development of active support tools.

A primary use of the electronic desk will be as an interface to advanced operational decision-support tools. Our development of support tools is predicated upon the design of active rather than passive aids. If a system is provided with limited knowledge about an office's organization and activities, including the specific identity of types of decisions to be made and their information requirements, it can exhibit "intelligent" behavior in support of the office worker. Its capabilities may include management of complex communications processes and other activities extended over time; generation of reminders, alerts and warnings; provision of unsolicited but relevant information; and detection of errors and anomalies. Bookkeeping support, convenient access to information, and active assistance will enable decisions to be made in a more timely, informed, and organized manner.

The second major area of our research uses the environment provided by the integrated system as a context in which to automate office procedures. We are engaged in the development of methodologies and tools for the construction of automated office systems, based on knowledge about the organization and function of an office. We feel that this knowledge can best be expressed in terms of a non-procedural problem-oriented specification language. A language of this type provides a user with both a conceptual framework for approaching the analysis of office functions, and a set of high-level structures natural to the problem domain in which to specify them. J. Kunin has begun work on the design of this facility, known as the Office Specification Language (OSL).

The design of OSL is based on the premise that there is a high degree of commonality of structure among the various procedures performed by a wide variety of offices, operating in seemingly disparate application environments. By identifying major types of activities carried out in offices, we have begun to abstract a set of semantic constructs which will evolve into the primitives of the language.

The specification of office procedures involves knowledge of three types: data, organization, and function. OSL provides constructs for describing the form and

structure of data used in the office, the inter- and intra-office organizational structure, and the specific functions for which the office is responsible. In the functional specification, the processing requirements are described in terms of a set of activities we have identified as canonical. These activities include structured processes, (such as management of communications, data management, logging and account reconciliation), as well as unstructured activities, or decisions, (such as scheduling, allocation, selection and verification). We are continuing the development of OSL, which we expect will be used as a framework and tool for formal analysis and communication of office procedures, and as an input language to an automation system.

Upon completion of the preliminary version of OSL, we anticipate building a prototype automation system. This will consist of several components. One will be a translator, which accepts OSL as input and transforms the specification into an internal representation suitable for expressing knowledge about the procedures and the data and organizational context in which to execute them. Another will be the program that interprets the internal representation. This system will keep track of when activities should be executed, automatically implement the structured activities as specified, and, maintain all bookkeeping needed to support human decision making.

Publications

1. Hammer, M. and McLeod, D, "The Semantic Data Model: A Modelling Mechanism for Data Base Applications." Proceedings of ACM SIGMOD International Conference on the Management of Data. Austin, Tx. June 1978.

2. Hammer, M. and Sarin S.K. "Efficient Monitoring of Database Assertions." to appear in ACM Transactions on Database Systems.

3. Hammer, M. "Very High Level Programming Languages." Proceedings of the Software Specification and Testing Technology Transfer Conference. Washington, D.C., Navy Laboratory Computing Committee and Office of Naval Research, April 1978.

4. Hammer, M. and Shipman, D. "An Overview of Reliability Mechanisms for a Distributed Data Base System." Proceedings of the 1978 Spring IEEE COMPCON Conference. San Francisco, Ca., March 1978.

5. Hammer, M. "The Impact of Data Management Research." Infotech State of the Art Report on Database Technology. London: Infotech, 1978.

6. Hammer, M. "The Impact of Automatic Programming Research." Proceedings of the 1978 National Computer Conference. Anaheim, Ca., June 1978.

7. Hammer, M., "Research Directions in Data Base Management." Proceedings of the Conference on Research Directions in Software Technology. Providence, R.I., Air Force Office of Scientific Research, Army Research Office, and Office of Naval Research. October 1977.

8. McLeod, D. "A Framework for Data Base Protection and Its Application to the INGRES and System R Data Base Management Systems." Proceedings of the 1977 IEEE COMPSAC Conference. Chicago Il., 1977.

9. Niamir, B. Attribute Partitioning in A Self-adaptive Relational Database System. M.I.T., Laboratory for Computer Science, LCS/TR-192. Cambridge, Ma., January 1978.

Theses Completed

1. Berkowitz, B. "Cost Models for the Datacomputer." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

2. Chuang, K.M. "On Representing the Distribution of Values in Data Bases." unpublished S.B. Thesis, M.I.T., Department Electrical Engineering and Computer Science, September 1977.

3. Danberg, S.A. "Evaluating Queries Concurrently in a Shared Database System." unpublished S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1978.

4. Karkula, S. "Information Flow in an Organization with Implications for Office Automation." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June, 1978.

5. Niamir, B. Attribute Partitioning in a Self-Adaptive Relational Database System. S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, January 1978.

6. Sarin, S. K. "Automatic Synthesis of Efficient Procedures for Database Integrity Checking." unpublished S.M. thesis, M.I.T., Department of Electrical Engineering and Computer Science, August 1977.

7. Shao, H. "Verification of the Structural Integrity of DBTG Databases." unpublished S.B. Thesis, M.I.T. Department of Electrical Engineering and Computer Science, May 1978.

8. Slutz, D. "Analysis of the Effects of Updates on Database Integrity." unpublished S.B. thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

9. Tuceryan, M., "Detection and Correction of Semantically Related Errors in Database Updates." unpublished S.B. thesis. M.I.T., Department of Electrical Engineering and Computer Science, June 1978.

10. Woltman, G., "Detection and Correction of Multiple Common-Site Errors in Database Updates." unpublished S.B. thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1978.

Theses in Progress

1. Chan, A. "A Methodology for Automating the Physical Design of Integrated Data Bases." Ph.D. Thesis, M.I.T. Department of Electrical Engineering and Computer Science, expected date of completion, August 1978.

2. Dell'Aquila, J.B. "Error Detection and Correction in Database Updates Using Imprecise Constraint Predicates." S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1978.

3. Koschella, J., "Some Optimizations of Nested Data Base Queries." S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, May 1979.

4. Kunin, J. "Analysis and Specification of Office Procedures." Ph.D. Thesis, M.I.T. Department of Electrical Engineering and Computer Science, expected date of completion, December 1979.

5. Leong, R. "Cost Minimization in Database Validity Checking." S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1978.

6. McLeod, D. "The Semantic Data Base Model and Its Associated Structured User Interface." Ph.D. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1978.

7. Wang, L. "Simultaneous File Partitioning and Index Selection in a Self-Adaptive Data Base Management System." S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1978.

8. Zdonik, S. "Semantic Query Optimization in Data Base Systems." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, May 1979.

Talks

1. Hammer, M. "Application Oriented Software Research." Conference on Research Directions in Software Technology, Providence, R.I. October 1977.

2. Hammer, M., "Research Directions in Data Base Management." Conference on Research Direction in Software Technology, Providence, R.I. October 1977.

3. McLeod, D. "A Framework for Data Base Protection and Its Application to the INGRES and System R Data Base Management Systems." IEEE COMPSAC Conference, November 1977.

4. McLeod, D. "Relational Data Base Management." California State University, Northridge, Ca. November 1977.

5. Hammer, M. "Model-Based Error Detection and Correction." NRL Workshop on Issues in Data Base Error Detection, Washington, D.C. January 1978.

6. McLeod, D. "The Semantic Data Base Model and Its Associated Structured User Interface." Presented at California State University, San Luis Obispo, CA, October 1977; University of California, Davis, CA, November 1977; California State University, Northridge, CA, November 1977; Rand Corporation, January 1978; Lawrence Berkeley Laboratory, Berkeley, Ca. January 1978; University of Southern California, Los Angeles, Ca. February 1978.

7. Hammer, M., "Very High Level Programming Languages." Software Specification and Testing Technology Transfer Conference, Washington, D.C. April 1978.

8. Hammer, M. "Applications Programming Methodology." Burroughs Corporation Computer Science and Technology Seminar, Nassau, Bahamas. April 1978.

9. McLeod, D. "Language Issues in Relational Data Base Systems." Boston Area ACM SIGPLAN Meeting, Boston, Ma. May 1978.

10. McLeod, D. "The Semantic Data Model: A Modeling Mechanism for Data Base Applications." ACM SIGMOD International Conference on the Management of Data, Austin, Tx. May 1978.

11. Hammer, M. "The Architecture of the Automated Office." M.I.T. Industrial Liason Program Symposium, Cambridge, Ma. May 25, 1978.

12. Sarin, S. K. "Efficient Monitoring of Database Assertions." 1978 ACM SIGMOD International Conference on Management of Data, Austin, Tx. May 1978.

DISTRIBUTED SYSTEM SEMANTICS WORKING GROUP

Academic Staff

B. H. Liskov,
Acting Group Leader
D. D. Clark
I. Greif
L. Svobodova

Graduate Students

A. Luniewski
D. Reed
E. Stark
E. Moss
C. Schaffert

Support Staff

V. Chambers
A. Rubin

DISTRIBUTED SYSTEM SEMANTICS WORKING GROUP

A. INTRODUCTION

Preliminary work on the structure of distributed systems, reported in last year's progress report in the section on the Computer Systems Research Group, resulted in the formation of a working group, composed largely of people from the Computer Systems Research Group and the Programming Methodology Group. This is the report of that working group.

Computer systems should reflect the structure and needs of the problems to which they are being applied. For many applications, a distributed computer system represents a natural realization. For both technical and economic reasons, it is likely that for many existing applications, distributed computer systems will replace conventional computer systems built around a large central processor, and that new applications will emerge based on distributed information processing. However, before such systems are feasible, a better understanding of how to construct them is needed. Our project is aimed at providing this understanding.

The move towards distributed processing has become feasible mainly because of the rapidly dropping cost of computer hardware and the increasing power and flexibility of mini and microcomputers. The move toward distributed systems will be dictated, however, by their "naturalness," and by the many technical advantages they offer over centralized systems. These advantages include the following:

1. Availability

   Availability of information can be increased by replicating it at several nodes. This arrangement not only increases the access bandwidth to the information, but in case of a failure of one of the nodes or a failure of some communication link, the information remains accessible.

2. Protection

   Distributed systems provide a better environment for protecting information stored in the system and for coping with run-time errors resulting from hardware failures or residual design and implementation errors. These advantages arise from the actual physical separation of independent or loosely coupled computations and information that belongs to different users. The physical boundaries of individual nodes provide "firewalls" that (if properly designed) will prevent spreading of errors originating in a particular node to the rest of the system. Such boundaries can also be utilized to protect information stored at individual nodes from unauthorized access or modification by other nodes. As the most severe protection measure, a self-contained node can be guaranteed privacy during some sensitive operation by physically detaching it from the rest of the system.

3. Reduced Software Complexity

The physical separation of computations and information may lead to a reduction in software complexity. Also, distribution reduces the level of hardware resource sharing, and, consequently, will reduce the complexity of software for resource allocation, scheduling, and protection. Lower software complexity makes verification of design and implementation more feasible.

4. Expandability

As more users join the system or new services are added, it is not necessary to make any physical replacements in a distributed system. Rather, one or more new nodes need to be added to the system. Distributed systems can grow more gradually than systems with a large central processor.

Thus, there are many sound reasons why applications should be implemented as distributed systems. However, there are a host of unsolved technical problems in building and programming a network of minicomputers to give the appearance of a coherent system. The project discussed in this report is to develop an integrated programming language and operating system to support a well-structured design and implementation of distributed applications.

1. Distributed Systems of Interest

The area of "distributed systems" has become a popular source of systems research projects. It has also become an important term in marketing computer equipment. Unfortunately, because of this popularity, the terms "distributed systems" and "distributed processing" are frequently misused, often referring to such conventional concepts as remote job entry, use of terminal concentrators, or multiprocessor organizations.

The distributed systems considered in our project can be described loosely as organizations of highly autonomous information processing modules, called nodes, which cooperate in a manner that produces an image of a coherent system on a certain defined level. Autonomy is the key characteristic that eliminates most multiprocessor organizations from this class of distributed systems. Certainly, a distributed system has more than one processor, since it has at least one processor in each node. However, in a distributed system, the nodes are highly independent, each having its own primary memory, possibly even some secondary storage, and its own interface through which it communicates with its environment (e.g. user terminals, sensors). The individual nodes are connected by a communication network; the communication delay may be highly variable and unpredictable. The communication network might be a long-haul network such as the ARPANET [15], a local area network [2], or a suitable combination of these two types. Each node has access to its own memory only; that is, inter-node communication is possible only by explicitly exchanging messages, not through shared memory. Finally, physical (geographical) reorganization of the nodes and the communciation network is assumed not to impair the system's functionality; the only change might be in the system's performance.

2. Comparison of Our Approach with Related Work

The assumption of autonomy of the nodes that compose a distributed system is the most important ingredient that distinguishes our work. However, once autonomy is assumed, the next issue that arises is to devise techniques that permit the programs running on the autonomous nodes to communicate in a coherent fashion. We are aiming at a high level of coherence that is application-independent but permits communication among the nodes in application-oriented terms. This high level of application-independent coherence distinguishes our approach from other work that is based on the assumption of autonomous nodes. Most work has either provided a very low level of coherence (e.g. the ARPANET) or has provided coherence within a specific application (e.g. the NSW works manager [12]). There is some work related to ours in progress at Xerox PARC, but again this work is focusing on a very specific application--office automation.

The problem of simultaneous update, making an identical or a logically related change at several sites, has received considerable study [5,10,13,14,16,18, 21]. However, we remain unconvinced that a solution to this particular problem is crucial to our research. Rather, we view our system as providing an environment in which any one of several simultaneous update algorithms can be implemented as needed. This point distinguishes our work from SDD-1 [16], for example, since that project assumes a very particular technique for implementing simultaneous update. SDD-1 also makes very restrictive assumptions about the autonomy of the nodes of the system.

Distributed systems have only lately become a focus of programming language research. In the past, programming languages have mostly not addressed concurrent programs. More recent languages (e.g. Concurrent Pascal [1]) Modula [22]) have had features for concurrency, but within the context of a single processor: these languages are based on the assumption that programs interact through shared memory, which is not consistent with the concept of autonomous nodes with private memory. There is related work at Oxford [9], the University of Rochester [6] and at MIT [4,7], but this work does not place strong emphasis on integrating the language and operating system features.

Indeed, we feel that our emphasis on the integration of language and system is a key factor that distinguishes our work from other related work. Much of what distributed programs do, falls into what is usually considered to be the systems area, including such topics as synchronization of access to shared information and protection. However, programs are written in a programming language, and proper primitives in that language can greatly influence the structure of programs. By integrating the two areas we expect to achieve a greater impact on the construction of distributed systems than could be accomplished in either area separately.

B. STUDY OF APPLICATIONS

It is essential that the mechanisms we develop to support construction of distributed applications will cover the real distributed processing problems. To this end, we have studied a number of applications, both by direct observation [19,20] and by surveying related work as discussed earlier. This study was hampered by the lack of existing distributed systems; for example, banking systems are not yet distributed,

although a distributed system is being planned. Therefore, we had to supplement our study by sketching designs for future systems.

Several different classes of distributed activities have been identified:

1. Invocation of Remote Servers

   A message is sent to a remote node instructing some server at the node to perform a certain operation; a reply (requested information or an acknowledgement if no data is to be returned) confirms that the operation has been performed. The mail system in the ARPANET is an example of this type of application.

2. Atomic Transactions on Distributed Databases

   To preserve the integrity of a database, it may be necessary to provide a mechanism that guarantees that either all updates specified by a transaction will be performed, or none, no matter how the transaction fails.

3. Distributed Data Processing

   If large quantities of data residing at different nodes are processed, a problem may arise even if no updates are performed, which is to minimize the data moved between nodes in order to perform the desired operation. An example is query processing in a distributed database system.

4. Distributed Problem Solving

   This describes systems where the cost (overhead) of maintaining a centralized global view of the system state and control is prohibitive. In such systems, each node knows only a partial state of the system and has to make intelligent guesses about the rest of the system. An example of such an application is a dynamic routing algorithm for store-and-forward networks.

5. Distributed Programming System

   This is a distributed version of a general purpose time sharing system. The assumption is that it is not possible to restrict in advance the modes of sharing among users. It is necessary to communicate both data and programs, but from the point of view of the mechanics of the actual exchange of information this type of system could be included in the first category.

The distribution can take place along two main lines, based on functional separability or on the non-uniform distribution of the use of databases. Functional distribution means that different nodes support different services. Such systems seem natural for control of industrial processes, where different nodes control different parts of a process, or in such systems as aircraft, where different nodes process information from different sensors. However, this approach seems to be also advantageous in service sectors such as banking [19].

Another category of distributed systems is a system where an individual processor supports the same services but on a different part of a database. A typical example is a bank with many branch offices. Each branch has its local accounts, but it should be able to serve a bank's customer whose account is at another branch. Since such remote requests are much less frequent than manipulation of the local accounts, partitioning of the bank's accounts database (that is, maintaining accounts on a computer at their local branch) is a natural approach.

It must be said that the division between functional distribution and database distribution is not clean; in most cases, a distributed system will to some extent include both. The latter case, however, implies an integrated database, while in the former case (functional distribution) the databases used by individual servers are much more independent. In some ways, the functional distribution is a more general case. A distributed database represents a special problem, the need to enforce consistency constraints that span several nodes. It is not clear how often this problem actually arises, but it cannot be ignored.

It can be concluded that the basic paradigm in the class of distributed systems that our project is addressing is the invocation of remote servers. This can be viewed as a communication protocol of much higher level than, for example, the host-to-host communication protocols currently employed in the ARPANET. The implementation of such high level protocols, however, may need to differ, depending on the type of application, and possibly on the efficiency and reliability requirements of the application. Therefore, we should not aim to design such high level protocols, but instead develop a set of tools that facilitate design and implementation of such protocols.

Finally, an application study by d'Oliveira [3] revealed an important result that there are strong pressures toward decentralization for sociological and political rather than technical reasons. These non-technical pressures imply to us that decisions about the distribution of information among the various nodes will be made for external reasons that only the application itself can specify. Thus, the application builder must have control over and understand the placement of information.

C. THE TARGET OF THE PROJECT

As was mentioned earlier, we view a distributed system as a collection of autonomous nodes that only communicate by information exchange over the communication network that connects them. In such a system, at least two levels of coherence must be enforced. One level is the application level itself. The second level is the set of internode communication protocols that facilitate the physical exchange of information (packets of bits). But there is a large gap between the application and the low level communication protocols. Usually, this gap results in a rather ad hoc implementation of the application.

Our target is an intermediate level, called the programming system, which will support a well-structured design, implementation, maintenance and control of distributed applications. This level is more than a programming language in a traditional sense. Rather, this level is envisioned as a set of tools that include primitives found in conventional higher level languages such as Pascal or PL/1, but also primitives normally

assumed to be a part of an operating system, for example, long-term storage and cataloging of information or control of protection safeguards. Thus, this programming level will integrate the programming language and the operating system. More strongly, this level will integrate a programming language and a distributed operating system. The design goals for the programming system include:

a. Aim for as high a level as possible, but application independent. Our system is intended to be used to implement many diverse applications, for example, both command and control systems and administrative systems like inventory control systems. To adequately support such a class of applications, the language should be as high level as possible but general purpose. One need that all applications share is the ability to exchange potentially quite sophisticated messages.

b. Support well-structured programming. Since our primary motivation is to ease the task of the application programmer, we feel that the embedded language should borrow from existing language work, in particular building on languages such as CLU [11] and Alphard [23], which aid in the production of well-structured programs by providing powerful abstraction mechanisms. Of particular importance is the data abstraction, which consists of a set of objects together with a set of operations that provide the only means for manipulating those objects. Data abstractions have so far been investigated mainly in the context of centralized processing. We believe that they will be even more useful in distributed systems, because they provide a powerful tool in organizing a coherent structure for distributed systems by permitting the data of the application and the allowed distributed sharing to be described in application-oriented terms.

   Since we are dealing with a distributed environment where an operation defined on the application level may require the assistance of several nodes, the language must support concurrent activities (process abstractions). Extensions of sequential languages will be necessary to achieve this. To enhance ease of use, we will keep the language as conventional and conservative as is consistent with our other goals.

c. Support communication in terms of abstract objects. Automonous program units need to communicate in terms of the kinds of high level objects they manipulate. For example, the ARPANET supports one sort of "high level object," the ASCII file, but any other form of data must be transmitted as a sequence of bits and explicitly transformed from one representation to another by a user written program. The language should support communication via messages composed of abstract objects. Two advantages arise from this approach. First, a clear statement can be made about the properties of data that the units depend on. Second, the approach clarifies the processing that is needed to translate an object in memory into a message transportable by the communication network and vice versa: the translation is accomplished using special operations of the object's type. Note that this translation is always needed; a language that requires messages to be composed of low level objects simply obscures this fact.

d. Allow explicit control of the application distribution. Conceptually, the target level can be viewed as an abstract network of processes where application-defined processes communicate via messages that contain high level commands, data and responses. In an ideal situation, this is all that would need to be seen by the application programmer. However, underneath this abstract network is the set of physical nodes and the communication lines that connect them. Our study of applications has indicated that the mapping of the objects used by an application into the physical set of nodes has to be made visible to the application programmer. We are also assuming that objects do not move dynamically from node to node, depending on the degree of demand (such dynamic migration is often assumed in the "distributed" systems consisting of many, relatively tightly coupled, mini or microprocessors). Rather, when a specific node is chosen to be the (new) home of a particular object, an installation of the object has to be explicitly requested, using commands provided by the programming system. This assumption is based on the belief, discussed earlier, that such placement decisions will often be based on non-technical factors external to the system [3].

e. Support sharing. The programming level must support sharing of objects that reside at different nodes and belong to different users, where what objects can be shared is defined by the application. An important aspect of sharing is to provide controls that regulate the patterns of sharing so that protection and synchronization constraints are properly met. It is also necessary to solve problems of naming across nodes.

f. Support reliable (robust) operations. Reliability is one of the most important goals of our project. A distributed system, by its very nature, provides a potential for enhanced reliability. However, to exploit this potential, the system and the application have to be properly designed. An arriving message must be tested for integrity and authenticity, using a combination of automatic system features and application dependent procedures, and there must be control over timeouts and the number of retries for messages sent but for which a reply has not been received. It is also desirable to have a means of specifying that an online backup copy is requested for an object.

g. Support changing patterns of use. We cannot expect an application to be written once and never modified. First, the system will grow by the addition of new nodes. Second, new patterns of use will arise involving existing or new pieces of information. Thus, we can expect synchronization and protection constraints to change with time. This change must not cause upheaval in the design of existing parts of the application.

We want to emphasize that the envisioned programming system is not intended for the end user, but for the application builder (programmer), although in some environments (such as LCS) there is often little distinction between the two classes of users. Also, it should not be necessary for all nodes in the distributed system to support the full language; each node need only support the appropriate (high level) internode communication protocol.

D. ENTITIES

In this section we discuss the universe of entities (e.g. programs, data) that take part in a distributed computation. We are not concerned with all aspects of the behavior of the entities, but rather limit our attention to questions concerning the locations of entities within the network and the possible relationships among the entities. We assume that each entity has an identity that is permanent; an entity can be referred to by giving its name.

1. Location of Entities

The universe of entities is spread across the physical nodes that make up the network. One question that arises concerns the location of entities: is an entity permanently located at a particular node, or can it move from node to node?

To make a decision here, we must consider several issues:

a. Earlier we discussed our conclusion, based on an analysis of applications, that the application programmers must be able to control the location of entities. Note that, at the least, this conclusion precludes automatic relocation of entities by the system, although relocation under program control would still be possible.

b. We are assuming that nodes are autonomous and possibly heterogeneous. Even under program control it is possible to move an entity to an autonomous node only if that node is willing to accept it. Furthermore, if that node is different from the current home node of the entity, considerable translation may be needed to effectively move the entity.

Therefore, we believe that entities should have a permanent location at some node in the network. An entity comes into existence at some node (when it is created) and remains at that node until it is destroyed. Moving an entity can be accomplished by having a program create a new entity and letting it "take over" from the old one; however, the relationship between the two entities is not recognized by the system, and represents a higher level concept of identity than that introduced above.

One consequence of this decision is that it will be easy for the system to create unique names for entities and to interpret entity names, since the node of residence can be part of the name.

2. Types of Entities

One of the fundamental decisions in developing a model of computation is to determine whether the entities used in the model are all uniform or to determine whether there are different classes of entities. Basically, the uniformity concerns the ways in which the entities may be used (and may use other entities). An example of a system in which all entities are uniform is the Actor System [7]: here every entity is an actor, and an actor is used by sending it a message (which is also an actor).

We have chosen to have different kinds of entities in our model. At this level of discussion, we are interested in distinguishing only two kinds of entities: processes

and everything else. A process is active, and is thought of as being the execution of a sequential program. Non-process entities, which we will call objects, are passive, i.e., they do not originate any activity. Examples of objects are integers, arrays, stacks, procedures, etc. Objects have a state (value) that may change. If the state can change during the object's lifetime, then the object is mutable.

A process can communicate with another process by sending it a message. We assume that the syntax and semantics of message passing is independent of the nodes of residence of the two communicating processes (although certain optimizations can be performed by the system if both processes reside at the same node). A process can use an object by performing (invoking) an operation on it (or by invoking it if it is a procedure); again, the semantics of invocation is the same regardless of the nodes of residence.

We have just described a model in which there are two basic primitives: invocation and message passing. We intend that the semantics of invocation is distinct from message passing: the primitives are really different. (We expect that these two primitives will also be distinguished syntactically, but that is a separate decision.)

If an actor-like view is taken, there is only one basic primitive, message passing, so our model seems more complicated. However, we believe that it is more natural than the actor model and will therefore be easier for programmers to understand. If programs built out of actors are examined, it is clear that there are "data-like" actors, "procedure-like" actors and "process-like" actors. We believe these differences are fundamental and should be reflected in the language and its semantics.

3. Restrictions on Referring to Entities

Now we address the subject of entities referring to entities. An entity may refer to another entity by using or containing its name. For example, a process will have local variables that may contain the names of other entities (both processes and objects); as the process executes, it can use these names. A data object is represented by some storage (at its node of residence), and some of this storage may contain names of entities (again both processes and objects).

One possibility is to place no restrictions on what entities can refer to other entities. Thus, a process could perform an operation on an object whether that object resided at the same node or not. Invocation of an operation on a remote object can be made to work, but has a disadvantage in that what appears to be a simple invocation will involve internode communication, and therefore can take a long time (although the semantics is still that of invocation). (The invocation must take place at the object's node, since the object cannot move to the invoker's node.)

We have chosen to restrict the objects that a process can refer to such that:

a. All these objects are at the same node as the process.

b. These objects are private to the process: no other process can refer to them.

There are no restrictions on the processes to which a process can refer. It is easy to enforce the above restriction as follows: messages can contain the names of processes but not the names of objects. A model obeying this restriction is illustrated in Figure 1. The nodes labeled Pi are processes, while nodes labeled Oi are objects. Two kinds of directed arcs are shown. A solid arc from entity x to entity y means y is a process and x names y, while a dashed arc means y is an object and x names y. Note that objects can name both processes and objects.

A process may ultimately refer to an object in the course of its execution if there is a path from the process to the object consisting entirely of dashed arcs. We will call the set of objects that a process may refer to its local address space.

Note that in this model, processes are analogous to nodes of the network: each process has a private memory and can communicate with other processes only by sending messages. Thus the universe of entities represents an abstract network. (The Abstract network model is possible because we distinguish processes from other entities. We believe this is another reason why it is worthwhile to make the distinction.)

The abstract network model has several advantages:

a. The programmer organizes the locations of entities by considering where to locate the abstract nodes (e.g., each process with its local memory). This seems easier than worrying about each entity individually.

b. Operations are always invoked locally. This is simpler to implement than remote invocation, and also avoids some arbitrary time delays. (Of course, the operation invocation itself might send a message, e.g., to some process whose name was contained in the object.)

c. Management of storage for objects (e.g., garbage collection) can be done locally on each node.

Although two processes cannot refer to the same object, they can share an object if they both name the process that can refer to the object. Such a process is called a guardian; it may guard one or several objects. The job of a guardian is to synchronize possibly concurrent requests to perform operations on the guarded objects. In Figure 1, P3 is a guardian for O7, which is shared by P1 and P2.

A guardian should not be assumed to know a priori about all processes that may request operations on the guarded objects. Furthermore, if a process requests an operation on data that are available only through the guardian, such a request may fail, since the guardian may refuse to release requested data, or in some cases may even destroy the data at its own discretion.

The abstract network model requires two extensions to be useful. First, the requirement that local address spaces of processes are disjoint may need to be relaxed. To obtain sufficient parallelism, it will probably be necessary to support complicated guardians consisting of several processes that share objects. This could be

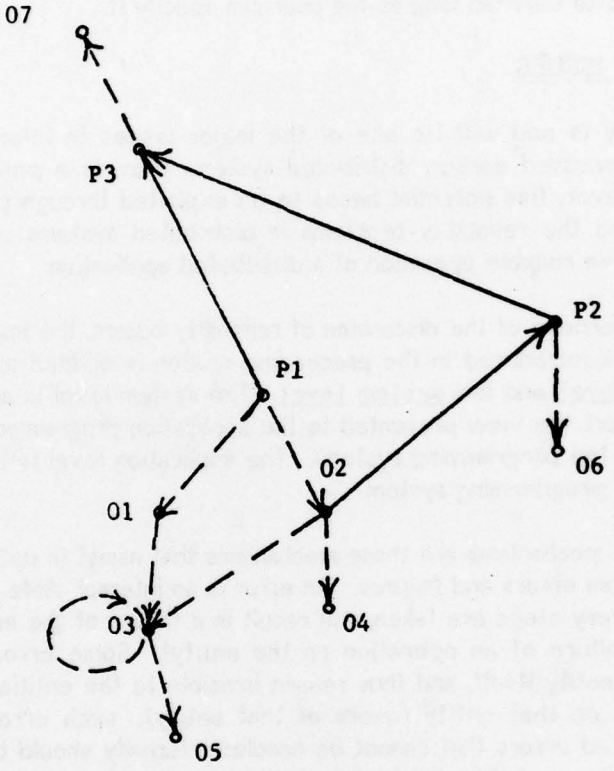


Figure 1. Example of Possible Relationship of Processes and Objects

accomplished by a special syntactic construct, something like a serializer [8], that defines the processes making up the guardian and their intercommunication; all the processes in the guardian would reside at the same node.

Second, in the case of a guardian that guards several objects, some efficient mechanism is needed that permits a user process to specify to the guardian the particular object of interest, and for the guardian to determine that the object so specified is one it guards. The system provides no guarantee, however, that such an object continues to exist as long as the user can specify it.

E. RELIABILITY ISSUES

Reliability is and will be one of the major issues in information processing systems. As discussed earlier, distributed systems provide a potential for enhanced reliability; however, this potential needs to be exploited through proper design. This section discusses the reliability problems in distributed systems and the mechanisms needed to achieve reliable operation of a distributed application.

For the purpose of the discussion of reliability issues, the implementation of the abstract network introduced in the preceeding section is divided into two levels: the application level and the system level. The system level is all the mechanisms needed to support the view presented to the application programmer (that is, the run-time support of the programming system). The application level is built using the tools provided by the programming system.

Reliability mechanisms are those mechanisms that assist in detection of, reporting and recovery from errors and failures. An error is an internal state of an entity that, if no special recovery steps are taken, will result in a failure of the entity (or, in case of data objects, failure of an operation on the entity). Some errors can be handled entirely by the entity itself, and thus remain invisible to the entities that are in some way dependent on that entity (users of that entity); such errors are said to be masked. Detected errors that cannot be handled internally should be reported to the users, by signalling a failure. Undetected errors turn into failures; it is possible that a user of such an entity can detect this kind of failure, but the problem is much more complex than with the reported failures.

To achieve reliable operations from the application point of view, both the system level and the application level have to include mechanisms for detection and handling of errors and failures. For simplicity, the following discussion uses the term failure to indicate reported and unreported failures as well as errors. For each type of failure, it is necessary to decide where it can be detected and how it should be handled. Some classes of failures, detected within the system level, can be masked, but for others a failure has to be reported to the application level. Some failures, however, are application dependent and therefore, their detection and handling has to be left to the application level. (In the class of system level failures, there is a gray area where a decision has to be made as to whether these failures will be masked by the system level or reported to the application level.) Thus, the system ought to provide sufficient mechanisms for masking certain classes of failures arising from the operation of the hardware and the software that supports the application programs. However, the system also has to provide suitable language constructs for the

application programmer to facilitate handling of the application specific failures and communication of the system detected failures to the application programs.

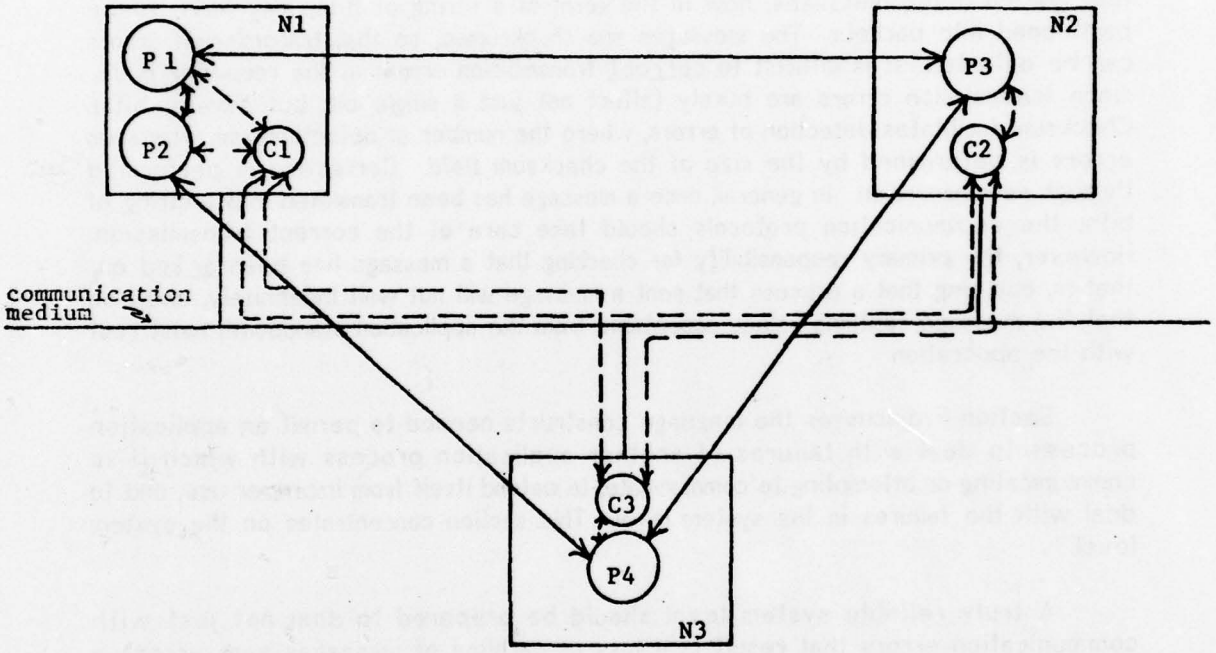
1. Communication Protocols

The abstract network is supported by a physical network of nodes and communication lines. Figure 2 shows the abstract network mapped into the physical network and the communication processes that control the physical delivery of messages among the nodes. The application processes exchange messages that, logically, contain values of abstract objects meaningful at that level. The values of these objects have to be translated or encoded into a string of bits for delivery to another node and decoded to the proper abstract objects at the receiving node. At the system level, messages, now in the form of a string of bits, may have to be partitioned into packets. The messages are checksumed, so that transmission errors can be detected. It is difficult to correct transmission errors at the receiving node, since transmission errors are bursty (affect not just a single bit, but several bits). Checksum facilitates detection of errors, where the number of detectable simultaneous errors is determined by the size of the checksum field. Correction is performed through retransmission. In general, once a message has been translated into a string of bits, the communication protocols should take care of the correct transmission. However, the primary responsibility for checking that a message has been acted on, that is, ensuring that a process that sent a message will not wait indefinitely, and also that the message contains values acceptable from the application standpoint, must rest with the application.

Section F discusses the language constructs needed to permit an application process to deal with failures of another application process with which it is communicating or attempting to communicate, to defend itself from improper use, and to deal with the failures in the system level. This section concentrates on the system level.

A truly reliable system level should be prepared to deal not just with communication errors that result in a loss or garbling of messages sent across a physical communication link. A reliable system level should not lose messages that have been presented to it by the application processes and queued for delivery. That is, the message queues should be recoverable in case of a physical failure of a node. This requirement becomes very important if translation from an abstract data object to the corresponding bit representation is a costly operation, or if the input to such a translation step is not automatically repeated (e.g. message typed by a user). This argument can be extended to the requirement that the system should guarantee delivery of all messages it has accepted from the application processes. That means that in addition to providing recoverable queues for messages that have not been sent yet, the system must continue trying to send the queued messages until it eventually succeeds. At the receiving node, the messages have to be stored again in recoverable queues, until they are picked up by the target application process.

Note that the recoverable queues and persistent retransmission could be pushed onto the application level. Putting it in the system level frees the programmer from some of the work needed to satisfy the reliability requirements and hopefully increases efficiency. However, the reliability mechanisms do represent potentially large



Pi application processes
Ni nodes
Ci communication processes
Solid lines: possible communications on the application level
Dashed lines: flow of information between processes

Figure 2. The Abstract Network: Communication on the Application Level and on the System Level

overhead, and their use should not be imposed on all communications. The basic communication scheme should be simple, fast, and inexpensive. We will investigate whether it is possible to vary the degree of reliability provided by the system by letting the application programmer choose from several different protocols; such protocols would be implemented as extensions (abstractions) built from the basic protocol.

2. Individual Objects: Correctness and Availability

In addition to the problem of communication, it is necessary to address the problem of reliability for individual objects in the system. This problem has three aspects:

a. no information should be accidentally lost or damaged

b. operations on objects should perform correctly (conform to the specification)

c. objects should always be available to qualified users (subject to protection constraints).

Redundancy is important for all three aspects. Two or more copies of an object stored and controlled in an independent way decrease the probability that the object will be lost or damaged as a result of a failure of the storage device or the processor manipulating the information. Also, if an operation on an object fails, it may be necessary to undo the effects of the operation (for example, an operation may have to be aborted because of a detected deadlock or because of a failure of some entity it uses). Redundant copies make it possible to restore the current state or to backup some earlier state of an object.

The issues regarding the reliability of individual objects are not specific to a distributed system; any information processing system should support backup and recovery of stored objects. Distributed systems, however, can increase the availability of information and services. "Availability" can be interpreted as the delay experienced when accessing a particular object. This definition has two connotations: one is the efficiency of the system, that is, the actual physical delay and queuing time in the abstract network (case E); the other source of delays are the failures in the abstract network, that is, the reliability aspects (case R). Redundancy is used for both of these subcases:

> Case R: If some particular node or communication with a particular node fails, it should be possible for the other nodes to continue their work. Since the failed (or inaccessible) node may contain objects needed by the other nodes, to increase availability means to maintain several copies of shared (shareable) objects on different nodes.
>
> Case E: Even if the system never fails, a single copy may not provide sufficient availability. A single copy of information or service may become a bottleneck; also, the communication delays, especially in a long-haul network, may be substantial, and it thus may be desirable to have a local copy (and, consequently, support multiple copies).

The question that needs to be answered is to what extent the individual copies have to be mutually consistent. It is important to distinguish between the two cases since the right solutions to the problem of mutual consistency are significantly different. In case R, only one copy needs to be actively used, that is, an object has a master copy and one or more backup copies. The changes made to the master copy must be propagated to the backup copies, immediately if every state of the object must be recoverable, or periodically upon special command if in a case of a failure it is sufficient to back out to some consistent state, not necessarily the last consistent state. In case E, all copies must be available for active use. It is often assumed that all copies must always be the same, but this requirement may defeat the very purpose for which the multiple copies were introduced: reduction of delays. The delay caused by synchronization of updates with other updates and accesses of multiple copies may exceed the delay that would result if only one copy were maintained. In case E it seems much more realistic to allow for multiple versions of an object; the local copy may not always be the most current version, but the most current version is known and a local copy of it can be obtained upon request.

The system level ought to support, in a selective way, the kind of redundancy required for case R. However, as discussed for case E, maintaining several equivalent copies of an object at different nodes can be expensive. One possible solution is to make individual nodes ultra reliable and use redundant communication paths between any pair of nodes. Selected objects then ought to be suitably replicated within a single node. For many applications, this may be the right approach. Case E is more complex and more application dependent. The application user, as well as the application programmer, may need to be aware that several versions of an object exist. Thus, case E should be left to the application level. A possible scenario is a system that on the application level supports multiple versions of selected objects, where the most current version is backed up on the system level.

F. LANGUAGE CONSTRUCTS FOR SENDING AND RECEIVING MESSAGES

An important issue in designing a language for distributed systems is how the language recognizes pairing of messages. The basic scenario in the abstract network is one process sending a message to another process requesting some action; later there should be another message, flowing in the other direction, indicating the result of the action. It must be possible to express in the language that the two messages are related. In addition, it is necessary to address the problem that the reply may never arrive, or that the request message cannot be sent. Several approaches are possible that differ in how long (for what event) the sending process must wait before it can proceed. Closely related to this degree of waiting is what kind of failures are detectable as part of the send command.

1. The Waiting Approach

In this approach, the sending process is forced to wait until the response comes back from the receiver, or some timeout or failure results. A possible syntax might be:

```
send C(args) to A timeout time:

    R1(formals) do S1;

    R2(formals) do S1;

    ...

    failure (formals) do Sfailure;

    timeout do Stimeout;

    end;
```

Here A is a process and C(args) is the message, consisting of a command, C, and some arguments. The remainder of the construct lists the various possible responses, together with the appropriate action to be taken by the sending process. R1, R2, etc., are responses for A; some might be normal, and some abnormal. "Failure" covers various failures that are detected either by the system or by the receiving process A. The arguments of "failure" specify the type of failure. Some examples of a detected failure are:

a. the message as specified cannot be constructed

b. the specified process (A) no longer exists

c. the target node is inaccessible

d. congestion (the target node or the target process (A) does not have enough buffer space)

e. the message cannot be decoded (the abstract objects contained in a message cannot be reconstructed).

Which of these failures are visible at the application level depends on the design of the system level. As discussed in Section E, the system level might be designed in such a way that message delivery is guaranteed. This would eliminate the need to cope with the failures of the type c and d at the application level.

The timeout action is taken if "time" is exceeded. If the system is designed for guaranteed delivery, the timeout action that terminates the send command should release the buffers in which the system keeps the message for delivery to the target application process. It should be understood that this timeout is for the pair of messages to be exchanged between processes in the abstract network; a different timeout value is used in the underlying system to govern retransmission of packets.

A different kind of "send" command is needed in the receiving process, since the receiving process must be able to respond to the command without waiting for the original sender process to respond back. To receive messages, A might use a construct:

```
command case
  ...
  C(formals) do...reply R(args);...;
  ...
end;
```

Here, A is waiting for one of a number of messages; if several are available, one is selected in a fair way. The message is then decoded, the contained data assigned to the formats, and the statements associated with the selected message are executed. The reply command sends a message back to the process that sent the message. Another form of reply:

```
reply R(args) to B
```

which explicitly names the process to reply to will probably also be needed. (This would permit a third process to be the replier to the original sender.)

The approach sketched above has the obvious advantage of pairing sends and receives. It also has some obvious disadvantages. For one thing, there are two send commands. More important, however, is the loss of parallelism. If the sending process had other tasks to do while its request was being processed, it must either not do them, thus reducing efficiency, or it must spawn another process to do these tasks. Thus a language supporting this approach must provide rich facilities for parallelism. (Note: this is not the only reason for which such facilities for parallelism might be needed. See the discussion of guardians in Section D.)

2. The No-Wait Approach

This approach is the opposite of the waiting approach: the sending process does not wait at all but continues running, performing actions on the local objects, or possibly even sending more messages. When it needs a response that is not ready, it waits for it. The language now has to provide additional constructs that allow the programmer to distinguish which response goes with which request and to specify that the process wishes to wait for the reply to a specific message, rather than a reply to any message that may have been previously sent.

There are various ways in which these problems might be solved. For example, send commands might be labelled:

S1: send C1(args) to A1;

S2: send C2(args) to A2;

...

get S1 replies:...;

In this approach, each send command has a continuation (as in Actors [7]) that can be named to identify the responses of interest in replies. Following replies would be the list of alternative responses, as shown in the preceding section, to the message sent by statement S1. Note that the errors arising in turning Ci(args) into a message would be exceptions (abnormal terminations) of the send command; failures such as (c), (d) and (e) described earlier would have to be reported outside of the send command (in replies).

Another possible approach is to use ports:

send C(args) to A reply-to P

where P is a port that can be named by more than one send command. Ports offer flexibility in expressing different patterns of requests and replies, both between a single pair of processes and in cases where a process communicates with several other processes. The port scheme could be further extended to allow the programmer to use a special port for replies indicating a failure:

send C(args) to A reply-to P failure-to F

The port F could be viewed as an entry to the "complaint department" of the respective application process.

The no-wait approach permits parallelism and is more flexible, especially in connection with ports. However, the linguistic mechanisms needed to enable the programmer to do the matching introduce extra complexity; how much flexibility is gained and how much complexity is added requires further study. The no-wait approach does not eliminate the need for supporting timeout, but now the timeout is specified at the point where the process must wait for the reply.

3. The In-Between Approach

This approach again makes the sender wait, but instead of waiting for the reply from the target process, the sender must wait only for some indication about the progress in the delivery of the message. For example, in Hoare's language [9], the sender waits until the replier receives the message.

The first question to ask is: does this approach offer the programmer any advantages over the other two approaches? Since sends and replies are not explicitly

paired, from this point of view the in-between approach offers similar advantages and disadvantages as the no-wait approach. What is gained over the no-wait approach is that certain failures, for example, (c) and (d), or possibly even (e) can be treated as exceptions of the send command. More importantly, the completion of the send command indicates that a meaningful message (to some extent) has been received, and, if the buffer into which the message has been placed is recoverable, it can be guaranteed that the message eventually will be processed. It is not clear, however, whether the guaranteed receipt and eventual processing of the message are really that important, or more precisely, whether it is important to know this right after the send command, instead of later when waiting for the results. It should be noted that there is a substantial loss of parallelism over the no-wait approach.

The in-between approach is often advocated on implementation grounds, as a means to prevent flooding of the receiver. Flooding means that messages are delivered faster than the receiver can process them. Since the buffer space of the receiver is always limited, either some control must be provided to stop the flow of messages or some messages must be discarded. In a system with shared memory, a very efficient implementation is possible, namely, each process has one send buffer, and the message is held there until the receiver wants it. In a system without shared memory such as our distributed system, this approach is clearly impractical, since extra messages would be needed to inform the communicating parties that a message is ready (sender to receiver) and that it can be transferred (receiver to sender). In a distributed system, the messages that cannot be processed immediately must be held not in the buffers of the sender but in the buffers of the receiver. Still, after a message from a particular sender has been discarded by the receiver for the lack of buffer space, the in-between approach can prevent the sender from sending additional messages. However, the flooding problem will be more appropriately handled on a lower level, especially in connection with the guaranteed delivery scheme.

G. PROTECTION ISSUES

In a distributed system, the protection problem can be simplified if we distinguish between inter-node and intra-node protection mechanisms. In the class of distributed systems considered in our project, a likely case is that a particular node is utilized by one user or at most by a set of cooperating and mutually trusting users. In this case, intra-node mechanisms are not required to have power sufficient to protect against subversion and malice. This is in strong contrast to a system such as Multics [17], and many other time-shared and multiprogrammed systems that were designed to operate properly with a set of mutually hostile users. The protection mechanism required in a single node is that which protects adequately against error and forgetfulness. This latter problem, while less severe than the problem that results from fully suspicious cooperation, is still not trivial. Presumably, the programmer must be provided a means of partitioning his computations, so that certain objects are accessible only in certain computations. This mechanism will allow him to debug new versions of software without running the risk of destroying existing objects.

We propose that a capability mechanism be the mechanism to provide this intra-node protection. By capability we mean an unforgeable identifier for an object, which identifies the type of the object. ("Capability" is often used to mean more than an unforgeable identifier: a capability may also include a specification of the access

rights, that is, a specification of which of the operations defined for the type of the object in question are actually allowed on that specific object. However, access control can also be achieved by making the object appear to be of the type that imposes the desired restrictions.) It must be presented as part of addressing an object. By constraining a procedure to execute with a limited collection of capabilities, it is easy to guarantee that the procedure will not do arbitrary damage to stored information.

Capabilities have certain disadvantages as a protection mechanism, but they are not apparent in this context. For example, inside a single node it should not be necessary to ask the question "Who are all the people who can get to this object?" This is a question that capabilities cannot easily answer. A properly designed cataloging mechanism will provide all the functionality in this direction that is required. Most importantly, the efficiency of capabilities becomes very important in this context in comparison to an alternative mechanism such as access control lists. Since we assume a world with a large number of small objects, it is clearly impossible to imagine that every object comes complete with an access control list; the overhead of an access control list for each object might be substantially larger than the object itself. Capabilities, on the other hand, need be no more than slightly enlarged addresses. We thus propose that the intra-node protection mechanism is based on capabilities, with some sort of capability cataloging mechanism playing the role now associated with the traditional file system.

Let us begin the discussion of inter-node protection by considering a point of policy rather than mechanism: the claim that protection between nodes should be based on an access control list mechanism rather than a capability mechanism. This claim is not based on difficulty of implementation; either mechanism can be imagined. Rather, it is based on our perception of the high level needs of distributed applications. A fundamental way to characterize the difference between capabilities and access control lists is that capabilities do not provide any easy answer to the question "Who are all the people who can get to this object?" while access control lists make it very difficult to ask the question "What are all the objects that I can get to?" If one considers real world protection problems, including those drawn from domains other than the computer domain, the more workable model of protection generally turns out to be that based on access control lists. While capabilities are often used in the real-world, the most obvious example being keys, the drawbacks are well known. Keys are subject to unauthorized duplication, loss, theft, etc. More relevantly, capabilities (or keys) do not provide a means to support accountability.

Both inter-node and intra-node protection requires partitioning of computations into non-overlapping access domains. The abstract network derived in Section D already provides such partitioning: each "node" in this abstract network has its own local address space inaccessible to other nodes. The decisions about intra-node and inter-node protection mechanisms can be extended to the abstract network: specifically, capability mechanisms will be used inside an abstract node, while access control lists will be employed for inter-node communication.

One of the basic goals of our project is to allow the application programmer to work with application-oriented entities. The same concern applies in the area of protection. That is, protection constraints should be expressible in application-oriented

terms. Powerful abstraction mechanisms and the concept of abstract nodes both contribute towards this goal.

Let us look now at the inter-node protection problem in the abstract network from a slightly different viewpoint. It will be a rare case where a request occurring between nodes consists of nothing more than the reading or writing of a single primitive object. In most cases, we can expect the request to be composed of an aggregate of reads and writes on various objects, which the requesting node views as atomic. This is generally referred to as an atomic transaction. The thing that must be protected from outside is the right to execute this atomic transaction. It is quite possible that the isolated reads and writes that are required as part of this transaction are not legitimate for the outside user except as a part of this or some other transaction. The classic example is where we are willing to release the average of a set of numbers, but not the numbers themselves.

Thus, we are faced with the problem of concisely expressing higher level protection constraints. One possibility, which we propose to reject quickly, is that the inter-node message may consist of an arbitrary algorithm, expre ed in terms of primitive read and write operations, and this algorithm will be examined and confirmed at the receiving node before execution to ensure that it conforms to the higher level security constraints. The construction of a verification algorithm that ensures that an arbitrary program conforms to one of the number of high level protection constraints would be a challenge to the most optimistic of the program verification researchers. Thus, we are led to the conclusion that the language in which an inter-node request is expressed must have primitives whose functionality closely matches the expressed protection constraints, so that it is easy to confirm that a proposed transaction does or does not fall within the bounds of the outstanding protection constraints.

We postulate the idea, common in data management systems, that different users of a data base have different views of the data base, often called different data models. From the outside, the data model appears to describe physically stored information and the acceptable operations on it. However, internally, the data model may have little correspondence to the information actually stored. Rather, it may be realized as algorithms that derive the modeled data from the information actually stored. Thus, we first see users being divided into large groups, based on which data model they use, and then being further divided within those groups according to which operations they can perform on the data model provided. For example, some users may be able to read certain records, others to read and write them. Each data model implies the existence of an algorithm to translate between that data model and the actually stored information. It is these algorithms that must be provided in advance, one set for each data model. The programming system must provide facilities for creating such data models, mapping them into the actual stored information, and synchronizing read and write operations originating from different data models.

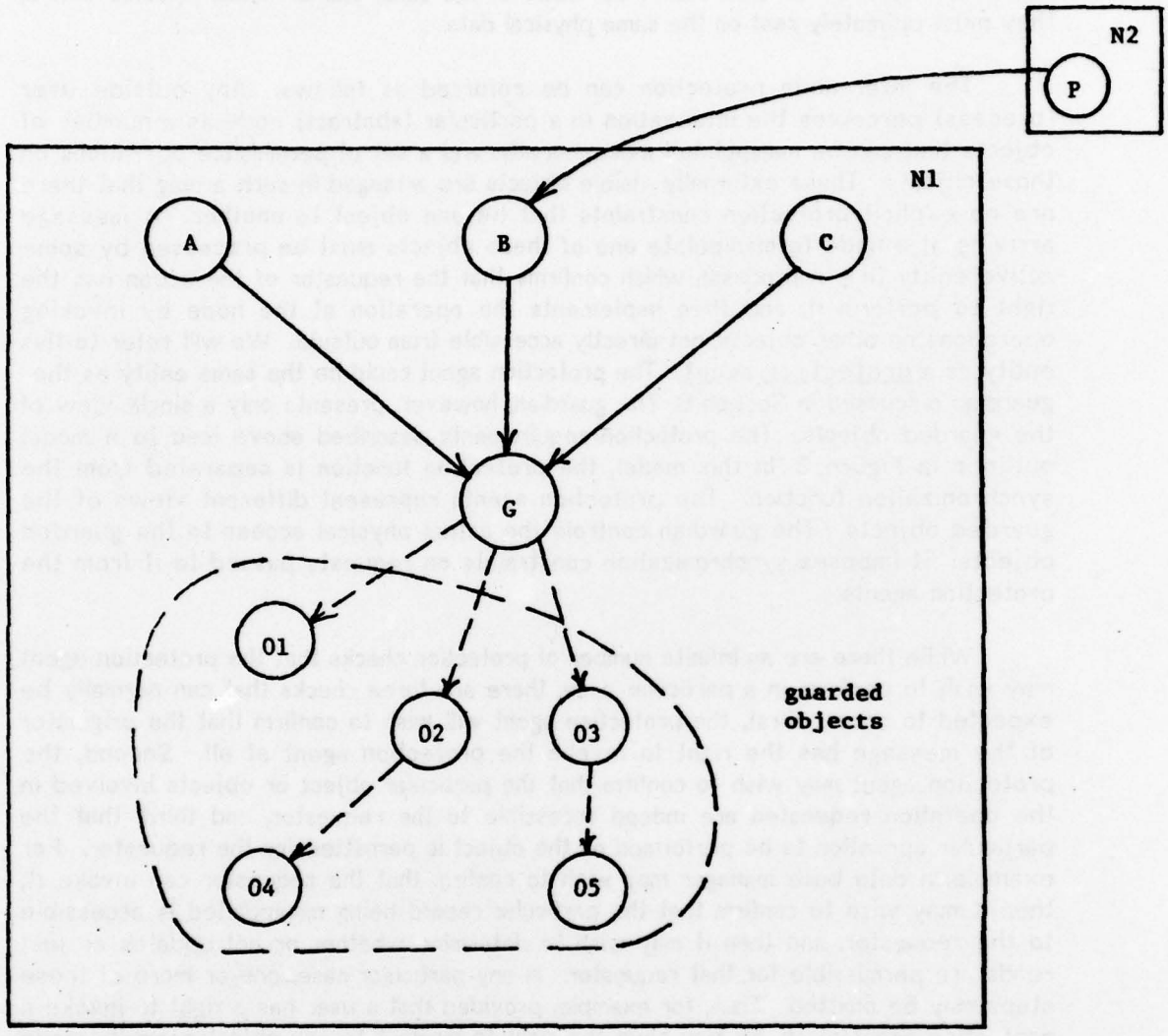
We have stated this paradigm in terms of the traditional vocabulary of data management. Let us state it again in a different vocabulary, that of typed objects. An abstract type, which allows only certain well defined operations on the objects of that type, while in reality it may perform arbitrary computation on a possibly large number of objects that constitute its representation, is very close to the idea of a data model. The traditional view of data models permits a low-level information entity to be shared

by different users through a variety of data models. To support this view via abstract types, it must be possible to manipulate a single low level object as part of a number of different abstract data objects, depending on the rights of the different users. The idea of data models is that different users have different views of the world, but, fundamentally, they do turn out to be views of the same world. Thus, in some sense, they must ultimately rest on the same physical data.

The inter-node protection can be enforced as follows. Any outside user (process) perceives the information in a particular (abstract) node as a number of objects that can be manipulated independently, and a set of permissible operations on those objects. These externally visible objects are arranged in such a way that there are no explicit protection constraints that tie one object to another. A message arriving at a node to manipulate one of these objects must be processed by some active entity (e.g. a process), which confirms that the requestor of the action has the right to perform it, and then implements the operation at the node by invoking operations on other objects, not directly accessible from outside. We will refer to this entity as a protection agent. The protection agent could be the same entity as the guardian discussed in Section D. The guardian, however, presents only a single view of the guarded objects. The protection requirements described above lead to a model outlined in Figure 3. In this model, the protection function is separated from the synchronization function. The protection agents represent different views of the guarded objects. The guardian controls the actual physical access to the guarded objects; it imposes synchronization constraints on requests passed to it from the protection agents.

While there are an infinite number of protection checks that the protection agent may wish to perform in a particular case, there are three checks that can normally be expected to occur. First, the protection agent will wish to confirm that the originator of the message has the right to invoke the protection agent at all. Second, the protection agent may wish to confirm that the particular object or objects involved in the operation requested are indeed accessible to the requestor, and third, that the particular operation to be performed on the object is permitted for the requestor. For example, a data base manager may wish to confirm that the requestor can invoke it, then it may wish to confirm that the particular record being manipulated is accessible to the requestor, and then it may wish to determine whether or not updates or just reads are permissible for that requestor. In any particular case, one or more of these steps may be omitted. Thus, for example, provided that a user has a right to invoke a protection agent at all, he may have the right to manipulate any object normally made accessible through that agent. The user may also be permitted to perform any of the operations defined on the objects. In that case, only the first of the three tests need be performed.

We should now pause and consider how this representation of protection meshes with the conclusion drawn earlier that inter-node protection should be expressed in terms of access control lists. Clearly, the use of access control lists implies that the protection agent must be able to reliably determine the originator of every message. Using the terminology developed for characterization of protection mechanisms in a centralized system, we will assume that every message, at its origin, has associated with it a principal identifier, which identifies the entity to be held accountable for the request in the message. Some techniques such as encryption



A, B, C: protection agents
G: guardian

Figure 3. Inter-node Protection Mechanism in Abstract Network: the Process P (Abstract Node N2) Can Reach the Objects Guarded by G (Abstract Node N1) Only Through the Protection Agent B

will be used to ensure the believability of the principal id by the recipient of the message, if the message has originated from a different physical node than the recipient; however, we will not describe such a technique here. Using this principal id, the first protection check described above is easy to implement. We can associate with every protection agent an access control list, and insist that the principal identifier associated with the message be on that list before the protection agent be invoked at all. The second test, that of ensuring that this principal is allowed to manipulate the particular objects in question, can be handled in a variety of ways. One obvious technique is to associate with each entry in the access control list, a list of all the objects that the particular principal is allowed to use. The protection agent can then refer to the list to determine the access privileges of the requestor. If the third type of protection check is required, it can be implemented as part of this same list, by associating with the entry for each object a notation describing the particular operations that this principal is permitted to perform on that object.

REFERENCES

1. Brinch Hansen, P. "The Programming Language Concurrent Pascal." IEEE Transactions on Software Engineering, Vol. SE-1 No. 2 (1977), 199-207.

2. Clark, D.D. et al. "An Introduction to Local Area Networks." M.I.T., Laboratory for Computer Science, Computer Systems Research Division, RFC-163. April 1978. DRAFT

3. d'Oliveira, C.R. An Analysis of Computer Decentralization. M.I.T., Laboratory for Computer Science, LCS/TM-90. Cambridge, Ma., 1977.

4. Dennis, J.B. First Version of a Data Flow Procedure Language. M.I.T., Laboratory for Computer Science, LCS/TM-61. Cambridge, Ma., 1975.

5. Eswaran, K.P. et al. "The Notions of Consistency and Predicate Locks in a Database System." Communications of the ACM, Vol. 19 No. 11 (November 1977), 624-633.

6. Feldman, J.A. A Programming Methodology for Distributed Computing. University of Rochester, Department of Computer Science, TR-9. Rochester, N.Y., 1977.

7. Hewitt, C. Viewing Control Structures as Patterns of Passing Messages. M.I.T., Artificial Intelligence Laboratory, A.I.M.410. December 1976.

8. Hewitt, C. et al. "Parallelism and Synchronization in Actor Systems." ACM Conference on Principles of Programming Languages. Los Angeles, Ca., January 1977.

9. Hoare, C.A.R. "Communicating Sequential Processes." Oxford University, University Computing Laboratory, Programming Research Group. Oxford, England. 1977, DRAFT.

10. Lampson, B., and Sturgis, H. "Crash Recovery in a Distributed Data Storage System." 1976. (To appear in Communications of ACM.

11. Liskov, B. et al. "Abstraction Mechanisms in CLU." Communications of ACM, Vol. 20 No. 8 (August 1977), 564-576.

12. Millstein, R.E. "The National Software Works: A Distributed Processing System." Proceedings of the ACM Annual Conference. Seattle, Wa., October 1977.

13. Montgomery, W. "Robust Synchronization in a Distributed Information System." M.I.T., Department of Electrical Engineering and Computer Science, Ph.D. Thesis *(in progress)*.

14. Reed, D.P. Naming and Synchronization in a Decentralized Computer System. M.I.T., Laboratory for Computer Science, LCS/TR-205. Cambridge, Ma., 1978.

15. Roberts, L.G., and Wessler, B.D. "Computer Network Development to Achieve Resource Sharing." AFIPS Conference Proceedings. Vol. 36, 1970.

16. Rothnie, J.B. et al. The Redundant Update Methodology of SDD-1: A System for Distributed Databases. Computer Corporation of America, Report CCA-77-02. Cambridge, Ma., February 1977.

17. Saltzer, J.H. "Protection and the Control of Information Sharing in Multics." Communications of the ACM, Vol. 17 No. 7 (July 1974), 388-402.

18. Stearns, R.E. et al. "Concurrency Control For Database Systems." IEEE Symposium on Foundations of Computer Science. Houston, Tx., October 1976.

19. Svobodova, L. "Distributed Computer System in a Bank: Notes on the First National City Bank." M.I.T., Laboratory for Computer Science, Computer Systems Research Division, RFC-155. Cambridge, Ma., January 1978.

20. Svobodova, L. "Distributed Computing in the Bank of America." M.I.T., Laboratory for Computer Science, Computer Systems Research Division, RFC-157. Cambridge, Ma., February 1978.

21. Thomas, R.H. A Solution to the Update Problem for Multiple Copy Data Bases which Use Distributed Control. Bolt, Beranek & Newman, Inc., Report No. 3340. Cambridge, Ma., July 1976.

22. Wirth, N. "Modula: A Language for Modular Multiprogramming." Software Practice and Experience, Vol. 7 No. 1 (January 1977).

23. Wulf, W.A. et al. "An Introduction to the Construction and Verification of Alphard Programs." IEEE Transactions on Software Engineering, Vol. SE-2 No. 4 (December 1976).

DOMAIN SPECIFIC SYSTEMS RESEARCH

Academic Staff

S. A. Ward, Group Leader
M. L. Dertouzos
P. Jessel

Research Staff

P. Houpt

Graduate Students

R. Archer
C. Baker
C. Cesar
J. Gula
R. Halstead
A. Mok
J. Pershing
A. Reuveni
B. Schunck
E. Strovink
T. Teixeira
C. Terman
J. Wahid

Undergraduate Students

Y. Gilbert
T. Hayes
R. McLellan
J. Sieber
S. Tomlinson

Support Staff

C. Eliot
N. MacKenzie
J. Pinella

DOMAIN SPECIFIC SYSTEMS RESEARCH

A. INTRODUCTION

Research of the D.S.S.R group during the past year has been directed toward the general problem of real time computation, with two major projects (CONSORT and the MuNet) emerging as foci for parallel research efforts which are expected to converge in the next year or so. Each of these projects has been described in previous progress reports; the following paragraphs serve to illuminate their respective current status and goals.

B. CONSORT: COMPILE-TIME TECHNOLOGY

CONSORT is a tool for the design of programs for a limited but important class of applications involving hard real-time constraints (e.g. control of physical processes or signal processing). During the past year an initial implementation of CONSORT has been completed by T. Teixeira, J. Pershing and A. Mok, and has been demonstrated in a "toy" application (balancing an inverted pendulum) by J. Wahid. A brief film of the demonstration is in preparation.

The current implementation translates a source program consisting of a block diagram (which may be input graphically) and produces an object program which runs on a single 8080 microprocessor. Static control structures are devised by CONSORT to meet real time performance criteria specified (in the source diagram) as latency constraints which dictate minimum rates at which data values must propagate through the diagram. A source program may consist of multiple pages, each corresponding to a particular control strategy . For example, object-time linkage mechanism provides for orderly transition between pages (passing state information to maintain continuity) as the target system passes from one phase of its operation to the next.

Experience with the current CONSORT implementation has been both encouraging and suggestive. Although many aspects of this initial effort are tentative and unpolished, the general approach it illustrates seems well suited as a basis for the construction of powerful engineering tools. In addition, the simple, very high level semantics of CONSORT programs together with specification of concrete performance criteria provide a nearly ideal context for the development of a variety of radical program transformation and optimization techniques. Current research by C. Terman and Teixeira explores such techniques and compiler organizations which exploit them, pursuant to a CONSORT reimplementation effort to begin in the next few months. Goals for the new implementation include:

1. Use of radical optimizations--e.g. substitution of table lookup for expression evaluation--to meet otherwise intractable real-time specifications.

2. Exploitation of more sophisticated (e.g. dynamic) control structures and multiprocessor target systems (ultimately, the MuNet). Relevent scheduling and partitioning problems are currently under study by Mok.

3. A variety of improvements in the human interface aspects of the system.

C. MUNET: OBJECT-TIME TECHNOLOGY

Our interest in multiprocessor systems is the potential they provide for a single, uniform target environment covering a wide range of the cost/performance spectrum. Such graceful scaling characteristics are particularly attractive in real time applications, where slight changes in a system's performance requirements may necessitate complete abandonment of a previous implementation in favor of a reimplementation on incompatible higher performance hardware. In addition, many "soft" real-time systems (e.g. for patient monitoring or industrial control) explicitly require extensibility over a wide range; as a result, such systems are typically over engineered so as to provide sufficient computation resources for the largest anticipated expansion.

Previously reported work by R. Halstead and S. Ward has led to a design for the MuNet, a multiprocessor architecture designed to address these issues. It consists of a sparsely connected network of small processors which support a message-passing protocol similar to (and inspired by) Hewitt's Actors. Noteworthy characteristics of the MuNet include:

1. At the lowest level, it supports a universe of data and code objects, each represented as a block of storage whose size may not exceed a system-wide constant. Thus higher level aggregate data must be represented as composites e.g. arrays of arbitrary size are represented as trees.

2. Neither objects nor object references imply absolute locality; thus in general, objects may be freely moved about the network (e.g. on the basis of dynamic load considerations).

3. The basic computation step (called an event) is time-bounded by a system-wide constant.

4. Each processor maintains an event list consisting of a FIFO queue of pending computations steps. The size of a processor's event list provides a first-order measure of the load on that processor; in particular, it provides a bound on the amount of time which may elapse before that processor will attend to a new event added to the bottom of the queue. This characteristic leads to interesting fairness and real-time properties.

5. Events may be moved between neighboring processors based on run-time load (e.g. to equalize event list sizes).

Implementation of a small (ten processor) prototype system is currently in progress. Major components of this effort include object management and scheduling (Halstead), compiler design and implementation (E. Strovink), operating system (J. Gula), and reliability measures (C. Baker).

A primitive version of the MuNet is expected to be operational during the Fall term, 1978, at which time various improvements and modifications will doubtless immediately suggest themselves. After a period of fine tuning, a variety of benchmark tests will be run to determine how various performance aspects scale with network size; we hope to find a nearly linear relationship over an interesting class of computations.

The absolute performance characteristics of this system will be of secondary importance, and may be less than impressive for several reasons. First, the generality and power of the MuNet environment stems in part from its heavy dependence on garbage-collected heap storage. Secondly, the partitioning of programs and data dictated by the system imposed time and space bounds introduce a certain amount of run-time overhead. Thirdly, little emphasis will be placed on the optimization of compiled code, at least initially. These factors may degrade performance by a factor of as much as ten or twenty in typical applications; thus the initial ten-node network may be outperformed in many cases by high-quality conventionally compiled code running on a single processor of comparable performance. This effect will be balanced by a compensatory increase in the flexibility and capacity of the multiprocessor system, and eventually should be largely mitigated by technical improvements.

Publications

1. Halstead, R. Multiple-processor Implementations of Message-Passing Systems. M.I.T., Laboratory for Computer Science LCS/TR-198. Cambridge, Ma., January 1978.

2. Teixeira, T. Real-Time Control Structures for Block Diagram Schemata. M.I.T., Laboratory for Computer Science LCS/TR-204. Cambridge, Ma., January 1978.

3. Terman, C. The Specification of Code Generation Algorithms. M.I.T., Laboratory for Computer Science LCS/TR-199. Cambridge, Ma., January 1978.

4. Ward, S. "Domain Specific Languages for Microprocessor-based Systems." Control Engineering, Vol. No. 24 (November 1977) 115.

5 Ward, S. "Approximate Contour Maps in Real Time." to appear in The Communications of the ACM, Vol. 21 No. 9 (September 1978).

Theses Completed

1. Gilbert, Y. "An Error Reporting Scheme for LR Parsers." unpublished S. B. Thesis M.I.T., Department of Electrical Engineering and Computer Science, June 1978

2. Mok, A. "A Frontend for a Morse Code Recognizer by Context." unpublished S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1977.

3. Mok, A. "Task Scheduling in the Control Robotics Environment." S. M. Thesis M.I.T., Department of Electrical Engineering and Computer Science, June 1977 (also, Laboratory for Computer Science, LCS/TM-77 Cambridge, Ma., 1976)

4. Pershing, J. "Design of A Domain Specific Metacompiler for Systems Using Graphical Input as a Source Language." unpublished S.M. Thesis M.I.T., Department of Electrical Engineering and Computer Science, January 1978.

5. Sieber, J. "A Packetized Communications Protocol for Remote File Access in the Laboratory." unpublished S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1978.

6. Wahid, J., "Implementation of Linear Quadratic Gaussian Compensators on Microprocessors." unpublished S. M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1978.

Theses in Progress

1. Archer, R. "Representation and Analysis of Real-Time Control Structures." S. M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, September 1978.

2. Baker, C. "Reliable Distributed Object Management Schemes." S. M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, May 1978.

3. Cesar, C. "Real Time Simulation Random Logic." Ph.D. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, May 1978.

4. Gula, J. "A Distributed Operating System for an Object Based Network." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, September 1978.

5. Schunck, B. "Analysis of the Effect of LQG Control on Computer Structures." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1978.

6. Tomlinson, S. "A Renaissance Machine Architecture." S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, December 1978.

Talks

1. Ward, S. "An Invitation to DELPHI." M.I.T. Alumni Seminar, Cambridge, Ma. July 1977.

2. Ward, S. "Towards a Renaissance Computer Architecture." MIDCON 77 Chicago, Il. November 1977.

KNOWLEDGE-BASED SYSTEMS

Academic Staff

W. A. Martin, Group Leader
L. B. Hawkinson
G. R. Ruth
P. Szolovits

Research Staff

G. P. Brown
G. S. Burke

Graduate Students

R. Baron
R. Bruccoleri

Undergraduate Students

J. H. Thompson

Support Staff

B. Demps
V. E. Lewis
R. E. Wagner

KNOWLEDGE-BASED SYSTEMS

A. RESEARCH SUMMARY

During the past year, our research was organized around three projects:

1. Knowledge Representation and Natural Language Processing--W. A. Martin, P. Szolovits, L. B. Hawkinson, R. Bruccoleri, J. H. Thompson

2. Natural Language Query to an On-line Dictionary--W. A. Martin, G. P. Brown

3. Very High Level Language Research--G. R. Ruth, G. S. Burke

B. KNOWLEDGE REPRESENTATION AND NATURAL LANGUAGE PROCESSING

The goal of this project is to create a programming system (OWL II) which will facilitate the creation of knowledge based application systems. The programming system is to include a general purpose natural language "front end." The major implemented components of this system are:

1. The Linguistic Memory System (LMS). This reads, prints, and manages a special data structure--Hawkinson.

2. A set of knowledge representation conventions and a grammar of English--Martin.

3. A parser--Szolovits and Martin.

LMS maintains a semantic net. Each node of this net is named by an expression formed by the binary combination of symbols and expressions naming other nodes. Each node has a reference (property) list. A decision was made to break this list into zones and this is being implemented. Simultaneously, LMS is being transfered to the MIT A.I. Laboratory LISP machine.

The grammar now covers about half the material in a typical English Handbook. We hope to cover all of such a handbook within the next year, making ours the most complete system. The grammar is specified using semantic nets and ATN networks, but differs from other systems in that production rules are used in the disambiguation of word and phrase senses. For example, for any transitive verb, v, if we can say "I v'ed the x," we can also say "x's v easily" e.g. "I killed the bug," "bugs kill easily." The formation of this latter sense from the former is specified in our system by a production rule. The rule is invoked by the parser whenever it is faced with an intransitive use of a transitive verb.

The parser is completely implemented with the exception of facilities for handling conjunction. The implementation emphasized clarity and flexibility rather than speed with the result that the average sentence takes roughly 10 cpu seconds to parse. Our current goal, for example, is to reduce this by an order of magnitude using

compilation. W. A. Woods achieved this sort of speed reduction on his system, but he had to reduce the coverage of grammar to do it. We hope to avoid this.

C. NATURAL LANGUAGE QUERY TO AN ON-LINE DATA DICTIONARY

This project is intended to apply the general facilities just described. In a large organization, people often have questions as to what data is available in computer analyzable form. This program will enter into a dialogue with a data base administrator in order to acquire a semantic level description of the organization's data bases. The system will drive the dialogue by menu selection. The system will then stand ready to answer users' questions about what data is available

D. AUTOMATIC PROGRAMMING

The general aim of automatic programming is to make the computer directly available to an end user with a problem to solve. The traditional method of system development proceeds roughly as follows:

Step 1: The user's desires and business are explained to a consultant.

Step 2: The consultant situation specifies the user's requirements for data processing.

Step 3: A software system analyst studies the specifications and fills in the remaining details.

Step 4: A programmer takes the completed design and writes a program that implements the design.

Step 5: A compiler processes the program and produces machine level code.

Obviously in this scenario the manager is far removed from the final result and the programmer is not generally cognizant of the real problem being solved. Every step adds to the time required to go from thought to action. Every step communicates imperfectly with its neighbors.

E. SCOPE OF OUR CURRENT WORK

The current ProtoSystem I project is concerned with the automation of steps 3 and 4. At step 3, one enters the realm of programming-language-like facilities. A specification is provided which, when fleshed out, serves as input for step 4. An analyst selects data structures, data access methods, and algorithms. The analyst gets size and access data and knows the performance requirements, then chooses an efficient design. The analyst does not, in general, create new data organizations, etc, but rather selects from those he already knows. Given the alternatives available to an analyst, choice criteria, and basic data about the system behavior, it is possible to automate the analyst function. An automated analyst should be more thorough and conscientious than a human one and much faster.

Step 4, from design to code, is the (relatively) straightforward process of

determining I/O and flow of control details and generating high-level code and JCL.

F. VERY HIGH LEVEL LANGUAGE DESIGN AND IMPLEMENTATION

Seeking entry into the software development process at step 3, and given the above view of that process, the ProtoSystem I project has evolved into the study of a Very High Level Language, HIBOL [HIgh level Business Oriented Language, also known as SSL (System Specification Language)], and an underlying automatic design and implementation system for it. [Needless to say, there have been a number of Very High Level Languages designed for business data processing, (e.g. the work of Hammer et al. at IBM Yorktown Heights, Prywes at the University of Pennsylvania, and Nunamaker). They vary in the detail and programming expertise that is expected of a user. The Very High Level Language systems also differ to the extent that automated design and code generation are performed. In fact, few systems provide these final steps or do so efficiently. The development of a Very High Level Language begins with a particular data processing domain and creates a language that captures and simplifies the important aspects of it so that its common data processing requirements are easy to describe. The language should allow only the functional specification of applications in the domain; ideally, the design and implementation is completed automatically. It is important that the language not allow too detailed a specification of processing and data. This would hamper the flexibility of the later design process.

The HIBOL language serves the business data processing domain. The language is non-procedural, "gotoless," and is not universal. It provides a stylized way of specifying file-oriented batch-processing systems.

A major concern of the VHLLs at the internal level is modelling of and proper handling of data sets with missing records. HIBOL provides a concise and powerful way of dealing with data aggregates. The language has a single data type, the flow. This construct is a conceptual data aggregate and represents a collection of uniform records that are individually and uniquely indexed by a multicomponent key. Each record has a single data field in addition to the key information. Scalar operators, "+," "*," etc. can be applied between flows. This causes the operation to be successively applied to all corresponding records (those with the same indices) of the argument flows. The result is a new flow which can be named or used as an intermediate value. There is a conditional value operator (similar to a "CASE" statement) which applies its tests and then performs the selected value expression as the individual records are processed. The primitive predicates allowed are the algebraic ordering predicates, ">," "=," and "<." The usual composition of these primitive predicates is allowed.

There is a class of reduction operators permitted on flows and flow expressions. Each (n-element) key is considered as a point in an n-dimensional space. A flow with keys in a smaller dimensional space is produced. All records of the input flows which project onto a single output record are "folded" into an accumulator so that the maximum, minimum, count, or sum of the projected records can be taken.

The restriction of a single datum per flow record allows the flows to be treated much like arrays in some programming languages. In particular, the treatment of operations on flows is similar to those operations on arrays in APL, although there is no explicit control of iteration details accessible to the designer in HIBOL. (There is also a mechanism for describing report contents and formatting in HIBOL.)

A major accomplishment of our research this year was the formalization by R. Baron of the semantics of HIBOL in his master's thesis. He begins by defining the BASBOL (Basic business Oriented Language), a semantically clean and explicit language for defining the semantics of a class of languages that manipulate data files in a batch processing environment. The language has a single data type, the *flow*, which is an aggregate of several data, each with identical structure. Each datum contains a *value field* and an *index* which uniquely identifies the datum. The single executable statement in BASBOL is the assignment statement, which specifies how to generate a single flow from other flows. Arithmetic, logical and various reduction, injection and projection operators provide the basis for the processing of the indexed data in flows.

For reasons of simplicity in reading and writing, the HIBOL language includes a number of defaults concerning the treatment of missing data. Although every attempt was made to make the resultant semantics natural, the potential for ambiguity existed. Baron was able to use BASBOL to resolve all potential ambiguities in a satisfactory way and to define the semantics of HIBOL cleanly and rigorously. This not only proves the integrity of the language, but provides the basis for experimentation in the development of extended and/or alternative semantics, as well.

G. DATA PROCESSING SYSTEM DESIGN

ProtoSystem I consists of four major modules. [A more detailed and comprehensive overview of the system is given in the 1976 Progress Report and in TM-70 (Ruth).] Control passes successively through three; the fourth is a utility. A Parser accepts HIBOL statements and produces "first cut" computations and "first cut" data sets. The Structural Analyzer (a utility component) models the properties of these primitive entities. The Optimizing Designer arranges and combines computations into programs and data sets into multi-field data sets. It also determines data set organization, blocking factors, key sort order, and access technique. These are chosen to provide minimal overall run-time cost. The Structural Analyzer is continuously used by the Designer to model the properties of the proposed computations and data sets. Lastly, the finished design of computations and data sets is passed to a PL/I Code and JCL Statement Generator.

The greater part of our research effort over the last few years has been addressed to the optimization of data processing system designs. Designer programs employing varying strategies and techniques have been written by Kornfeld, Alter, and Morgenstern, all aimed at minimizing the costs of programs produced from HIBOL descriptions. Program cost is dominated by read/write costs. The optimizers reorganize the programs within the confines of the generalized computation and data set formats. As indicated in the previous paragraph, these designers deal with gross program and data organization, not with such traditional issues as common subexpression elimination or strength reduction. The savings produced by the latter

techniques are insignificant compared to I/O costs. [For an amplified view of the issues and techniques involved in design and optimization see TM-72 (Ruth).]

Two questions immediately arise concerning any optimizing designer: (1) does it produce correct designs (i.e. does it preserve the semantics of the HIBOL program) and (2) how well does it optimize? We have taken great pains to ensure and verify the accuracy of our Designer, but assessing the quality of its optimization has been a more difficult problem. To be sure, we have verified that it performs the more obvious efficiency enhancements, but further investigation was deemed necessary.

To this end we have developed a "manual Designer" this year that allows the user entry to the software development somewhere between steps 3 and 4. This designer provides an Implementation Specification Language (ISL) for specifying all the details of an implementation that have been mentioned above. Using this tool and a translator that transforms Designer output to ISL we can now easily vary the automatically produced designs for experimental purposes. As an aid to the human user the manual Designer checks the consistency and completeness of the ISL design. This has also proven useful in further verification of the automated Designer's accuracy.

The data driven nature of computations provides an important opportunity for optimization. A computation's iteration through its input data sets is performed for some critical set of keys. Through structural analysis each collection of input data sets that encompasses this set of keys is calculated. The optimizer must choose the collection that minimizes auxiliary data set reads and that can be organized as a sequential and sorted data set. Through the use of Baron's semantic analysis of HIBOL and experimentation with the manual Designer it was found that the automatic Designer was too conservative in some cases, selecting an unnecessarily large number of driving data sets, and thus producing suboptimal designs. We have been able to make significant improvements to the optimizing Designer using the results of this investigation.

H. AUTOMATIC CODE GENERATION

Step 4 of the software production process is automated by the PL/I and JCL Code Generator. This module adds the implied I/O access routines and loop iteration control to the assignment statement expansion dictated by HIBOL to produce a number of finished programs. In the code expansion, it is imperative not to cause "READ" operations for data that might not be needed. I/O is a bit tricky because for some access routines, the support code may be in different places or even distributed across several places. The data driven character of the loops introduces some sequencing problems. The main one is ensuring that records in a data set not having all possible key tuples are not read in the wrong order.

Extensive manual verification of the PL/I and JCL Code Generator had been performed and the time came for machine verification. This year we tested the code produced for syntactic correctness through compilation on the UCLA 360/91 OS-MVT system. We have also begun further testing by running the compiled code on that system with sample input files.

KNOWLEDGE-BASED SYSTEMS GROUP 82

Publications

1. Brown, G. P. "Failure Handling in a Dialogue System." Proceedings of the Fifth International Joint Conference on Artificial Intelligence-Vol. 1. M.I.T., Cambridge, Ma., August 1977. Pittsburgh, Pa.: Carnegie-Mellon University, 1977.

2. Mark, W. "The Reformulation Approach to Building Expert Systems." Proceedings of the Fifth International Joint Conference on Artificial Intelligence-Vol. 1. M.I.T., Cambridge, Ma., August 1977. Pittsburgh, Pa.: Carnegie-Mellon University, 1977.

3. Martin, W. A. "Comment following article by Schank and Lehnert." Research Directions in Software Technology. Edited by Peter Wegner. Cambridge, Ma.: M.I.T. Press. To appear.

4. Martin, W. A. "Descriptions and the Specialization of Concepts." Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Vol. 2. M.I.T., Cambridge, Ma., August 1977. Pittsburgh, Pa.: Carnegie-Mellon University, 1977. Also available as M.I.T., Laboratory for Computer Science, M.I.T./LCS/TM-101, Cambridge, Ma., March 1978.

5. Martin, W. A. "Remarks on Knowledge-Based Programs." Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Vol. 2. M.I.T., Cambridge, Ma., August 1977. Pittsburgh, Pa.: Carnegie-Mellon University, 1977.

6. Ruth, G. R. "Automatic Programming: *Automating the Software System* Development Process." Research Directions in Software Technology. Edited by Peter Wegner. Cambridge, Ma.: M.I.T. Press, 1978.

7. Ruth, G. R. "Automatic Programming, A Survey." Proceedings of the Annual Conference of the Association for Computing Machinery. Seattle, Wash., October 1977.

8. Ruth, G. R. "Protosystem 1: An Automatic Programming System Prototype." Proceedings of the National Computer Conference, 1978. Anaheim, Ca. Montvale, N.J.: AFIPS Press, June 1978.

9. Swartout, W. R. "A Digitalis Therapy Advisor with Explanations." Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Vol. 2. M.I.T., Cambridge, Ma., August 1977. Pittsburgh, Pa.: Carnegie-Mellon University, 1977. Also available as M.I.T., Laboratory for Computer Science, M.I.T./LCS/TR-176. Cambridge, Ma., February 1977.

Theses Completed

1. Baron, R. B. "Structural Analysis in a Very High Level Language." unpublished M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, September 1977.

Theses in Progress

1. Bruccoleri, R. E. "English Conversational Error Correction in a Natural Language Parser." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, September 1978.

LOCAL AREA NETWORK WORKING GROUP

Academic Staff

D. D. Clark, Acting Group Leader

Research Staff

K. T. Pogran

Graduate Students

S. Kent
A. H. Mason
A. Mendelsohn
D. Reed

Undergraduate Students

R. Baldwin
H. Carter
N. Chiappa
C. Hornig
J. Maloney
T. McMahon
S. Ratliff
C. Schieck
A. Urbino

Support Staff

P. Baskin
O. Feingold
V. Newcomb
J. D. Ricchio
M. Webber

LOCAL AREA NETWORK WORKING GROUP

A. INTRODUCTION

The LCS Local Area Network project is a joint effort of the Computer Systems Research group and the Technical Services group. To those who have observed progress on the LCS Network only through these annual reports and other written communications from the Laboratory, it may seem that little progress has been made in 1977-78. In our previous annual report, for example, we stated that "the first three nodes on the net are expected to be operational within the next two months," and this most certainly did not happen. One year later, we state again that we expect to have a small, initial network in operation within a few months.

The appearance of little progress during the past year is deceiving, for as will be detailed below, it has been a year of great activity, from which we have learned a lot. Perhaps the most significant lesson learned, which we commend to anyone interested in implementing a local area network in the near future, is this: the technology of local area networks is not "off-the-shelf," neither the hardware nor the software for a high-bandwidth general purpose local area data communication network is available in a form in which it can be procured, installed, and be immediately operational. Organizations interested in installing local area networks should realize that today, at least, implementation of a local area network entails development efforts in both the hardware and software domains.

Lest this sound like too dire a prediction for the field of local area networking, let us add some rays of hope: interest in the field is growing, and expertise in it is following close behind, not only at the Laboratory for Computer Science, but at other centers as well. Most importantly, development efforts, both here and elsewhere, are gaining momentum. It is our goal to bring the design of the Local Network Interface, the hardware base for the LCS Network discussed below, to a point where it can be manufactured as a product. Whether or not it becomes available as a commercial product, its design will be in the public domain.

The effort that has gone into the development of the LCS Network falls into two broad categories, hardware development and software development. We shall now examine the progress that has been made in each of these two areas during the past year.

B. HARDWARE

As was mentioned in the CSR section of our last annual report, the primary hardware component of the LCS Network is a device called the Local Network Interface, or LNI. The LNI was designed by a group at the University of California at Irvine headed by D. Farber, under contract to DARPA. Use of the LNI as the hardware base for the LCS Network was attractive to us for several reasons. First, although it was specifically designed to control a ring network patterned after the Distributed Computing System ring network previously developed by D. Farber at UC-Irvine, its internal structure was general enough that it could be

modified to control a cable packet broadcast contention network, exemplified by the Ethernet developed at the Xerox Palo Alto Research Center. This flexibility would enable us to easily implement subnetworks of each type as part of the overall LCS Network, and would further our goal of comparing the two networks under operational conditions. Second, development of both the LNI and the LCS Network was funded by DARPA; it thus made a good deal of sense to join in the development of one hardware device, rather than have two devices produced through separate efforts. Third, the Laboratory for Computer Science, traditionally a software-oriented laboratory, wished to avoid the major hardware development effort that would be necessary to implement our own network hardware.

The estimate made in last year's annual report that the first few network nodes would become operational early in the 1977-78 year did not pan out. The first Local Network Interface arrived in October, rather than in mid-summer as had been anticipated. More significantly, it arrived essentially undebugged; K. Pogran tackled what turned out to be a major hardware debugging effort in conjunction with M. Lyle of the University of California at Irvine. Laboratory involvement in LNI development led to the creation, in January, 1978, of a new Technical Services group within the Laboratory, headed by K. Pogran, and the investment in a hardware development capability which the Laboratory had hoped to avoid, had, instead, been made.

The first two Local Network Interfaces were essentially operational by the end of May, connected as peripherals to the same PDP-11/40 for checkout purposes, communicating over what amounted to a two-host ring network, and properly performing hardware-level network communication functions. Communication between two PDP-11's using the same two LNI's is to be demonstrated during June, with delivery of a third up-to-date LNI expected at the end of the month.

Further hardware projects outlined in last year's annual report still remain to be done. These include development of a version of the LNI to interface to the Laboratory's PDP-10's and DECSYSTEM-20, and addition of "packet buffers" to the LNI to facilitate use of much higher network transmission rates than the 1Mb/s currently employed, and to facilitate interfacing of the LNI to lower-speed computers such as microprocessor-based systems. Also, a major redesign of the LNI is contemplated, which will include: design improvement in all aspects of the LNI; "modularization" to enable interchange of major components of the LNI, such as the host interface (PDP-11 DMA, PDP-10 I/O Bus, etc.), Name Table associative address store, and packet buffers; modification for control of an Ethernet-like network, to be implemented as another module of the LNI.

A few words should be said about the complexity of the current LNI. In its present form, the LNI comprises approximately 350 TTL SSI and MSI integrated circuits, apportioned as follows:

LNI "proper"	120
PDP-11 full-duplex DMA	100
Name Table Controller	25
Name Table Cells (8)	90
Test and diagnostic	15
Total	350

Assembled, the cost of each interface is approximately $2,500:

Wired backpanel, chassis, etc.	$1,700
Integrated Circuits	450
Miscellaneous parts	50
Final assembly, checkout, etc.	300
Total	$2,500

The LNI is of a complexity that, once its design is stable, it could reasonably be implemented as a single LSI "chip," or perhaps, at most, two chips. This could be done either via a funded research project, or at the initiative of an LSI manufacturer who sees a market for it. Certainly, the potential is there; the basic design concepts of the LNI are sound, and the day of local area networks is just dawning.

C. SOFTWARE

The delay in the availability of working Local Network Interfaces for the LCS Network has had both positive and negative impacts upon the implementation of software for the Network. Primarily, the software involved is the low-level or end-to-end communications protocol software which must be implemented on all LCS Network hosts. The delayed arrival of the LNI's has had a negative impact in that there has been no hardware for which to write and debug device drivers and similar software for the various machines and systems which will be part of the Network; it has had a positive impact in that it has given us more time to pursue the convergence of the Data Stream Protocol (DSP) initially designed for the LCS Network with the Transmission Control Protocol (TCP), the internetworking protocol.

We reported in last year's annual report that DSP was the end-to-end protocol of choice for the LCS Network, but that we were "involved in an effort to bring DSP and TCP together again, since TCP is the ARPANET standard for end-to-end communication in the 'internet' environment." This effort placed us squarely in the fray of internetwork protocol development; D. Clark and D. Reed have attended TCP Working group meetings, and D. Clark has become involved in other activities of the Internetwork Working group.

The Data Stream Protocol had its roots in TCP I, the original internet protocol. DSP was a "leaner" protocol than TCP I, providing the same functionality with a simpler structure. It was intended to be less cumbersome, and better suited to the high bandwidth environment of a local area network. However, because the LCS Network will not exist by itself in a vacuum, but will instead be interconnected to the ARPANET and, through it, to other networks, its protocols must be aware of and be capable of dealing with the internet environment. Thus, TCP and the LCS Network protocol must somehow mesh.

The result of a year's work by the TCP Working group is TCP III, an improved protocol for internetworking which was strongly influenced by DSP. Though TCP III is not the ideal protocol for a local area network, it is a reasonable protocol, and in the interests of compatibility with the internet environment, we have adopted it for use with the LCS Network. DSP, then, has served as a "straw man" that has helped TCP to evolve.

Implementation of TCP III has begun at LCS, as well as at other internet locations. By May of 1978, implementation of TCP III was underway for both the UNIX and Multics systems. There are two UNIX systems at LCS. One, on a PDP-11/40, will serve as the prototype of the LCS Net-ARPANET gateway; the other, operated by the Domain Specific Systems Research group, is a PDP-11/70 which will be a major host on the LCS Network. The TCP implementation for UNIX is based on an ARPANET NCP implementation for UNIX done at the University of Illinois and the Rand Corporation. The Multics system run by M.I.T.'s Information Processing Services will initially not be on the LCS Network, but with its TCP implementation, it will be able to communicate with LCS Network hosts via the ARPANET and the LCS Net-ARPANET gateway.

Theses Completed

1. Ratliff, Steven. "A Dynamic Routing Algorithm for a Local Packet Network." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, February 1978.

2. Urbina, Alejandro. "Performance of a Terminal Concentrator Under a Data Stream Protocol." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, December 1977.

PROGRAMMING METHODOLOGY

Academic Staff

B. H. Liskov, Group Leader

I. Greif

Research Staff

R. W. Scheifler

Graduate Students

R. R. Atkinson
V. A. Berzins
T. Bloom
D. Kapur
V. Ketelboeter
M. S. Laventhal
J. E. Moss
R. N. Principato
J. C. Schaffert
R. W. Scheifler
L. A. Snyder
M. K. Srivas
E. W. Stark

Undergraduate Students

S. C. Garrard
P. Leach
E. R. Schienbrood
C. M. Tan

Support Staff

S. Barefoot

A. L. Rubin

Visitors

R. Bergeron

A. Merey

PROGRAMMING METHODOLOGY

A. INTRODUCTION

The research efforts of this group are directed at developing tools and techniques for simplifying the production of software that is not only reliable, but easy to understand and maintain as well. At the center of our work is a programming methodology we have developed, whereby programs are constructed through a top down decomposition process driven by the recognition of abstractions [1]. A major focus of our research in previous years has been the design of a programming language-system, CLU, which supports this methodology; CLU provides powerful abstraction mechanisms that we believe are well matched to the task of building quality software.

This past year we have concentrated on writing both formal and informal definitions of CLU, to ensure that all features of the language and their interactions are well understood; we also worked on the implementation of CLU. We have continued our study of specification and verification techniques for programs based on data abstractions. In addition, we have explored the semantics and implementation of a CLU-like language that does not require a heap and garbage collection, and we have developed a method for automatically synthesizing synchronization code from specifications.

B. CLU DEFINITIONS

With the design of CLU essentially complete, we have turned to the task of writing complete definitions, both formal and informal. A description of the CLU exception handling mechanism, which supports the construction of fault tolerant programs, is given in [2], along with a discussion of the many design issues involved. In addition, a new version of the CLU Reference Manual is now nearly complete [3].

Two formal definitions of CLU have been made. J. C. Schaffert [4] has given an operational semantics using a new semantic method that he has developed, and R. Scheifler [5] has finished a denotational semantics using the Scott-Strachey approach to language definition. This research has provided us with the opportunity to evaluate various features of CLU from a new viewpoint. Although our understanding of the meaning of CLU programs has not really changed, several places were discovered where our understanding of what constitutes a legal program was faulty or incomplete, and changes were made to CLU as a result. Schaffert's work has demonstrated the usefulness of his technique for a non-trivial language, and Scheifler's definition is being used on an informal basis to verify the correctness of the legality-checking portion of the current CLU compiler.

C. CLU IMPLEMENTATION

As reported in last year's progress report [6], we are currently engaged in a second implementation of the CLU compiler and system, motivated chiefly by the desire for a relatively efficient and transportable implementation. The new implementation runs on a DEC PDP-10 under the ITS operating system, and we are in the midst of bringing up a version to run under TOPS-20.

At present there are three major components to the CLU system. The CLU compiler translates source text into a machine independent macro language called CLUMAC. The CLUMAC assembler, run automatically under control of the compiler, turns this intermediate text into binary code. A third program, CLUSYS, is used as the execution environment for CLU programs, and contains a loader, support routines, debugging facilities, and a simple expression evaluator.

Both the compiler and the assembler are written almost entirely in CLU itself. Most of CLUSYS is written in a mixture of CLUMAC and PDP-10 assembly language, and at present must be assembled using the standard ITS assembler, MIDAS. However, we are upgrading the CLUMAC assembler to handle "hand coded" as well as compiler generated text, so that our dependence on MIDAS can be eliminated.

An important difference between the new implementation and the previous one, in which the intermediate language was MDL (a high level, LISP-like language), is the way in which parameterized modules are implemented. Below, we first briefly describe parameterized modules, and then turn to their implementation.

1. Parameterized Modules

In CLU, procedures, iterators, and clusters can all be parameterized. Parameterization provides the ability to define a class of related abstractions by means of a single module. Parameters are limited to just a few types, including integers, strings, and types. The most interesting and useful of these are the type parameters: objects in CLU can grow and shrink dynamically, so size parameters are not needed.

When a module is parameterized by a type parameter, this implies that the module was written without knowledge of what the actual parameter type would be. Nevertheless, if the module is to do anything with objects of the parameter type, certain operations must be provided by an actual type. Information about required operations is described in a where clause, which is part of the heading of a parameterized module. For example,

```
set = cluster [t: type] is create, insert, delete, elements
                where t has equal: proctype (t, t) returns (bool)
```

is the heading of a parameterized cluster defining a generalized set abstraction. Sets of many different element types can be obtained from this cluster, but the where clause states that the element type is constrained to provide an *equal* operation.

As a second example, the parameterized procedure in Figure 1 defines a class of summing functions for collections (such as sets and arrays) of integers.

```
sum = proc [struc: type] (s: struc) returns (int)
        where struc has elements: itertype (struc) yields (int)
        x: int := 0
        for elt: int in struc$elements(s) do
                x := x + elt
                end
        return (x)
        end sum
```

Figure 1. Example of a parameterized procedure.

The **where** clause constrains the legal actual type parameters to those having an *elements* iterator of the appropriate type.

To use a parameterized module, actual values for the parameters must be provided, using the general form

module_name [parameter_values]

Parameter values must be computable at compile-time. Providing actual parameters selects one abstraction out of the class of related abstractions defined by the parameterized module; since the values are known at compile-time, the compiler can do the selection and can check that the where clause restrictions are satisfied. The result of the selection, in the case of a parameterized cluster, is a type, which can then be used in declarations and operation names; in the case of parameterized procedures or iterators, a procedure or iterator is obtained, which is then available for invocation. For example, *sum*[*set*[int]] is a use of the two abstractions shown above, and is legal because int provides an *equal* operation and *set*[int] provides an *elements* iterator.

2. Implementation

There are a number of basic schemes for implementing parameterized modules. These schemes can be characterized by the time at which the binding of actual parameter values takes place. The possible times include compile time, load time (after compilation but prior to execution), and run time (either at the first use of each distinct set of parameter values, or at every use). The result of binding parameters is called an *instantiation*.

In a compile-time binding scheme, the compiler produces a distinct object module for each distinct set of parameter values; each use of a formal parameter in the source text is replaced by the corresponding actual parameter, and then the resulting text is compiled to obtain the instantiation. In the load-time and run-time schemes, a parameterized abstraction is compiled into a single, parameterized object module; this module is later instantiated by supplying actual values for the parameters.

The compile-time scheme is similar to macro processing, and has many of the associated advantages and disadvantages. Its primary advantage results from the greater context that is available to the compiler when compiling any particular instantiation of a parameterized abstraction. This increased context allows the generation of more time-efficient object modules, both because of the greater

opportunities for optimization and because run-time binding is avoided. The primary disadvantages of this scheme are the increased number of compilations performed and the increased amount of space needed to store the object modules.

In the load-time and run-time schemes, binding is performed on object modules. The binding does not require that a new copy of an object module be created for each set of parameter values; rather, the code of the module and most of its local data can be made independent of the particular parameter values, and thus can be shared by the various instantiations.

There are two possible run-time schemes. In the first, the binding of parameters takes place each time a parameterized object module is invoked. The parameter values are passed to the object module as extra, hidden arguments, and are referred to by the object module just like the normal, explicit arguments. This was the scheme used in our previous implementation of CLU. In the second scheme, which is the one used in the current CLU implementation, a new object module is created once for each distinct set of parameter values; the binding occurs at the first use during execution. (Alternatively, one could run through storage looking for uses of parameterized modules and force binding to take place before execution.) The new object module is created by building a new structure containing the parameter-dependent data; the code of the module and its parameter-independent data are shared by the various instantiations.

Compile-time and load-time schemes all require that every possible set of parameter values supplied to an abstraction be determined before execution begins. In CLU, the possible parameter values are restricted to "compile-time computable" constants. However, despite this restriction, it is possible to implement recursive parameterized abstractions that use an unbounded number of distinct parameter values, as the following perfectly legal module (inspired by [7]) demonstrates:

```
agen = proc [t: type] (n: int) returns (any)
       if n <= 0
           then return (array[t]$new ())
           else return (agen[array[t]] (n - 1))
           end
       end agen
```

An invocation *agen*[*T*](*n*), where *T* is an arbitrary type, eventually produces a new array. The important characteristic of *agen*, however, is that *agen* calls itself recursively with a parameter array[*t*] that is distinct from the original parameter *t*; in fact, it is distinct from any previous parameter to *agen* within a single recursive chain of calls. For any positive *n*, an invocation of one instantiation of *agen* will use *n* distinct additional instantiations of *agen*. For example, the invocation *agen*int](3) will result in 3 recursive instantiations of *agen*:

```
agen [array [int]] (2)
agen [array [array [int]]] (1)
agen [array [array [array [int]]]] (0)
```

Thus there exist finite CLU programs that use at run-time an unbounded number of instantiations of parameterized abstractions. To handle such programs, it is therefore necessary to support the dynamic instantiation of parameterized abstractions at run-time. For a compile-time scheme to be correct, one must recognize modules such as *agen* and either consider them to be illegal, or provide some means for implementing them that avoids compiling an infinite number of object modules.

As was mentioned above, the current CLU implementation utilizes a run-time scheme wherein a new object module is created once for each distinct set of parameter values. Since in the implementation there is no single object module for a cluster as a whole, but rather individual object modules for each cluster operation, the following (somewhat simplistic) description focuses on the representation of routines. Types are represented, by objects called *type descriptors*; however, type descriptors are used primarily in various forms of identification, and their internal format is not of particular importance here.

The implementation makes use of two types of objects, *call blocks* and *entry blocks*. A call block is a description of a routine to be invoked, and contains the routine name, the actual parameters for the routine and a type descriptor for the data type, if the routine is a cluster operation. An entry block represents an invocable entity (i.e., a non-parameterized routine or an instantiation of a parameterized routine); it contains references to constituent objects containing the code for the routine, the parameter-independent data, and the parameter-dependent data. The parameter-independent data consists of literal values, such as real numbers and strings, and call blocks for invoked routines that are not dependent on the parameters. There is parameter-dependent data only in entry blocks for instantiations; this data consists of the actual parameters and call blocks for invoked routines that depend on those parameters.

For example, Figure 2 shows the entry block for the instantiation *sum*[*set*[**int**]]. This entry block refers to one parameter-independent call block, for **int$***add*, and one parameter-dependent call block, for *set*[**int**]**$***elements*. Notice that in the call block for *set*[**int**]**$***elements* there are no routine parameters; this is because *elements* has no parameters besides those of its containing cluster. A call block for *sum*[*set*[**int**]] is shown in Figure 3. Note that here there is a routine parameter, but no type descriptor, since *sum* is not an operation of a cluster.

The uninstantiated form of a parameterized routine is also represented by an entry block, to be used as a template when building instantiations. In the parameter-dependent data of this entry block, each would-be reference to the *ith* actual parameter is instead a reference to a dummy descriptor for "the *ith* parameter." For example, the template for *sum* looks like Figure 2, except that references to *set*[**int**] are replaced by references to "the first parameter."

Whenever an attempt is made to invoke a routine through a call block, a dynamic linker intervenes. If the entry block for the specified routine already exists, the call block is replaced by that entry block, thus *snapping* the link. If the entry block does not yet exist, i.e., a parameterized routine is being instantiated with a new set of parameters, a new entry block must first be created from the template entry block for the routine. The new entry block shares the code and the parameter-independent data with the template (and all other instantiations), but has a completely new copy of the

parameter-dependent data in which every reference to a dummy descriptor for "the *ith* parameter" is replaced by a reference to the corresponding actual parameter.

It is important to realize that instantiation merely involves substituting actual parameters into the parameter-dependent data template; no attempt is made to simultaneously snap the call blocks in the resulting data. One reason for this is that attempts to instantiate certain routines (such as *agen* above) would cause an infinite number of subsidiary instantiations. A second reason is that some (possibly many) of the call blocks may never be used, so snapping them is a waste of time. For example, code to handle potential, but unexpected, exceptions may never be executed.

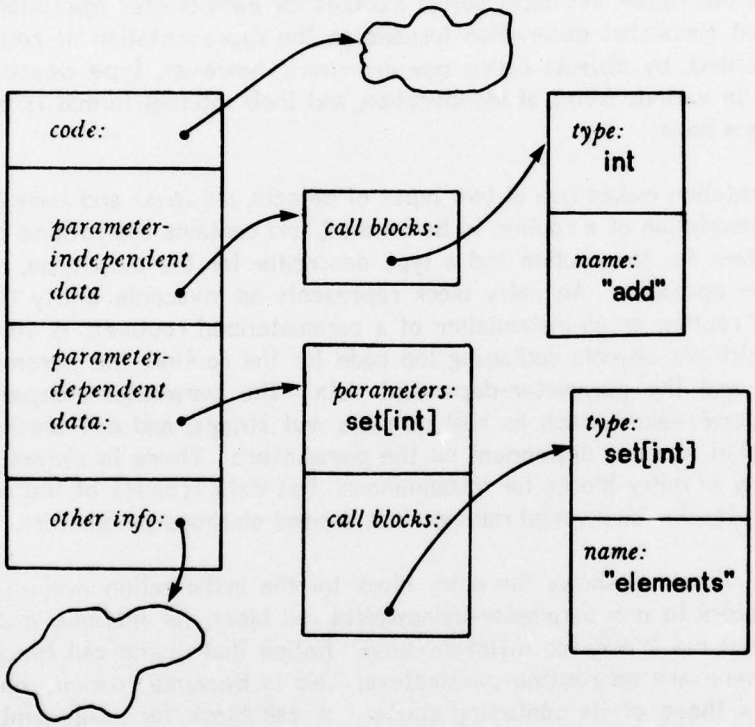


Figure 2. Entry block for *sum*[*set*[int]].

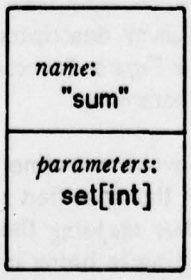


Figure 3. Call block for *sum*[*set*[int]].

The above description omits a number of details that are largely related to aspects of performance. For example, the parameter-dependent data in an entry block is actually separated into two parts: data dependent solely on cluster parameters, and data dependent on routine parameters (and perhaps also on cluster parameters); in this way, all operations of a parameterized type can share that data dependent on just the cluster parameters, while those (rare) operations that are additionally parameterized have separate, additional data dependent on those parameters. Although these details are important to the actual implementation, they do not fundamentally alter the description just given, and so will not be pursued here.

D. SPECIFICATION AND VERIFICATION OF DATA ABSTRACTIONS

We have continued our work on specification techniques for data abstractions. R. Principato has completed a formalization of the state machine technique [8], which is based on the work of Parnas [9] but makes use of hidden functions to define delayed effects of operations [10, 11, 12]. V. Berzins [13] is completing a formalization of the abstract model specification technique [14, 15]; his technique is powerful enough to permit the specification of mutable data abstractions and operations that raise exceptions. D. Kapur [16] is completing a formalization of the algebraic technique, using a model theoretic approach that permits the relationships between existing methods [17, 18] to be elucidated.

E. INCORPORATING ABSTRACT DATA TYPES IN STACK-BASED LANGUAGES

E. Moss has investigated how abstract data types might be included in a programming language based on stack rather than heap implementation [19]. The decision to use a heap is fundamental to CLU, and we felt that it contributed greatly to the simplicity of the language. However, there is a demand for languages supporting abstract data types without requiring garbage collection (e.g., the DOD/1 language [20]). Our goal was to investigate a possible design for such a language. CLU was used as the base for the alternative design, not only because it is a complete language, but also because it serves as a basis for comparison and evaluation of the resulting design.

In the sections below, we discuss the major decisions in the alternative design.

1. Basic Semantics

The semantics of CLU rests upon the fundamental notion of an object, and the secondary concepts of variables and assignment. Objects are abstractions of memory. Objects reside in a universe of objects; they may be created freely, and continue to exist as long as they are accessible. Objects may refer to other objects, and general sharing and cycles of references are permitted. Some types of objects (e.g., arrays) may grow and shrink dynamically. A heap implementation with some form of garbage collection is required to support the full semantics of CLU.

CLU variables merely refer to objects. In most cases variables are implemented as pointers to the storage representing the object to which they refer. Assignment copies only the reference, not the object, and hence affects no objects. Argument passing to procedures is defined in terms of assignment: the formal arguments are

assigned (references to) the actual argument objects. Thus, the new procedure activation shares objects with its caller. (This is not the same as passing a reference to a variable, which is never done.) We call this argument passing technique call-by-sharing.

These basic notions of object, variable, assignment, and argument passing need to be changed to permit stack implementation. First, variables are changed to be cells physically containing the objects to which they refer. This ties the storage and lifetime of objects to that of variables, which are allocated on the stack. Next, components of aggregates (such as arrays and records) are changed to be physically contained in the aggregate object, rather than merely pointed to. However, objects continue to play an important role in the semantics. For example, an array variable contains an array object, and is not a collection of scalar variables. (We will come back to this point in Section E.3.) Thus there is no notion of sub-variables: either one assigns entire objects, or one uses operations of the type to manipulate and update objects of that type.

In the new design, assignment and argument passing are not defined as they are in CLU. Procedure invocation becomes the semantic base. Procedures take all arguments by reference, but there are two classes of arguments: input arguments and output (or result) arguments. Result arguments are variables that the procedure must write but cannot read (since they may be uninitialized). The two classes of arguments are separated in the procedure header:

```
p = proc (a, b, c: int) returns (m, n: int)
```

They are also separated in invocations, with the result arguments appearing to the left of the assignment symbol, e.g.:

```
x, y := p(t, u, v)
```

Thus, assignment is defined by procedures that write into their result arguments. The built-in types provide operations to copy an input object into any given variable, and all other assignments are built up from these operations. This definition of assignment avoids much unnecessary copying.

In addition to the form

```
<list of result variables> := <procedure invocation>
```

defined above, a definition is needed for forms such as

```
x := y
```

where y is a variable. For convenience, we define such forms to be equivalent to

```
x := t$copy(y)
```

where t is the type of y.

after the Distributed Computing System ring network previously developed by D. Farber at UC-Irvine, its internal structure was general enough that it could be

As in CLU, the semantics of expressions are defined in terms of procedure calls. Here, however, anonymous temporary variables must be created to receive the results of the expressions and pass them on. The process of creating temporaries can be viewed as a syntactic transformation, and so we can think of expressions as a convenient shorthand for a series of procedure calls. For example,

```
x := 2 * z + 5
```

is equivalent to

```
t1: int := int$mul(2, z)
x := int$add(t1, 5)
```

2. Size Parameters

Since variables, and hence objects, are to be allocated on the stack, the size of objects now becomes important. For example, in CLU a string variable may refer to strings of any length. This is because strings are allocated in the heap and variables merely point to them. In the new language, the length of strings becomes important, since variables must physically contain them. The obvious solution is to add appropriate parameters to each type, to specify the size information. However, since to many programs the exact size does not matter, it is desirable to have the ability to write modules that handle all sizes of objects of a particular type. Indeed, using the CLU parameterization features, this is entirely possible. However, CLU's parameter mechanism is oriented towards statically (compile-time) known parameters, and it is often desirable to put off size choices until run-time. Furthermore, objects of different sizes will be of different types.

Distinguishing types based on object size leads to some problems. One difficulty is that each type in CLU has a distinct set of operations. Consider the string$fetch operation, which returns the ith character of a string given the string and i as arguments. There will now be a different string$fetch for each size string, written string[n]$fetch where n is the size of the string. Even worse are binary operations, for example, string$lt (which compares two strings and returns true if the first lexicographically precedes the second); these operations now take two parameters--one for the size of each argument. It becomes very tedious and error prone to keep track of such parameters.

To solve these problems, we devised a size parameterization mechanism where size parameters did not determine type, and where size parameters could be specified at run-time. Some other considerations also influenced the design of the mechanism. It was clear that size information had to be associated with variables in order to determine storage requirements. However, the compiler introduces temporary variables for expression evaluation, as explained above. How is the size of a temporary determined?

The solution chosen was to have procedure headings specify their result sizes as a function of their input sizes. For example, the header of string$concat would specify that the size of its result string is the sum of the sizes of its two input strings. This works fine if the result argument is a temporary created by the compiler, but

what if the result argument is an already declared variable supplied by the user? It is undesirable to require an exact match in size; for example, in invoking string$concat, any variable big enough to hold the result should be acceptable. The solution here is to decouple the sizes of objects and variables by requiring only that the object fit in the variable. In the general case this requires a run-time check since the sizes of both objects and variables may not be known until run-time.

One might suppose that comparisons of size parameters would be the basis for run-time size checks. However, user-defined types lead to a problem: the user is allowed to define the concrete size parameters in terms of the abstract size parameters using arbitrary expressions. Thus the concrete and abstract size information may not be related in any simple way, and storage requirements may not even increase monotonically with the abstract size. We decided that a comparison of the size of the object and variable in terms of storage units (e.g., words) be performed. This is not an entirely satisfactory solution, although it is simple and efficient; this problem needs more work.

The syntax we chose for size parameters was the following:

<type-name> [<regular parameters>;<size parameters>]

In those contexts where a type is needed but no size information is required (e.g., as inputs to procedures), the size information is ignored and may be omitted. Means are provided for accessing the size parameters of input arguments, etc.

3. Access to Components of Objects

The design as described so far is sufficient for most purposes, but has a limitation. The semantics of procedure invocation imply that the objects returned in result argument variables are always newly created. Thus there is no way to pass a component of an aggregate object to a procedure; only a copy of it may be passed, since fetching the component is done with a procedure call, which necessarily creates a copy.

It would be possible to define access to record and array components specially to avoid this problem. However, such a solution would not generalize to user-written aggregate types (e.g., sets, lists). Therefore, we added a new kind of module, called a selector, for returning a component of an aggregate by sharing. Record and array component access is defined by built-in selectors, and users may define selectors for their own types in terms of these built-in selectors.

It should be noted that restrictions on user-written selectors need more investigation. For example, rules are needed that simplify aliasing prevention. Also, we imposed syntactic restrictions on the use of selectors to prevent dangling references (selectors effectively return a reference), but our solution is not as clean as we would like.

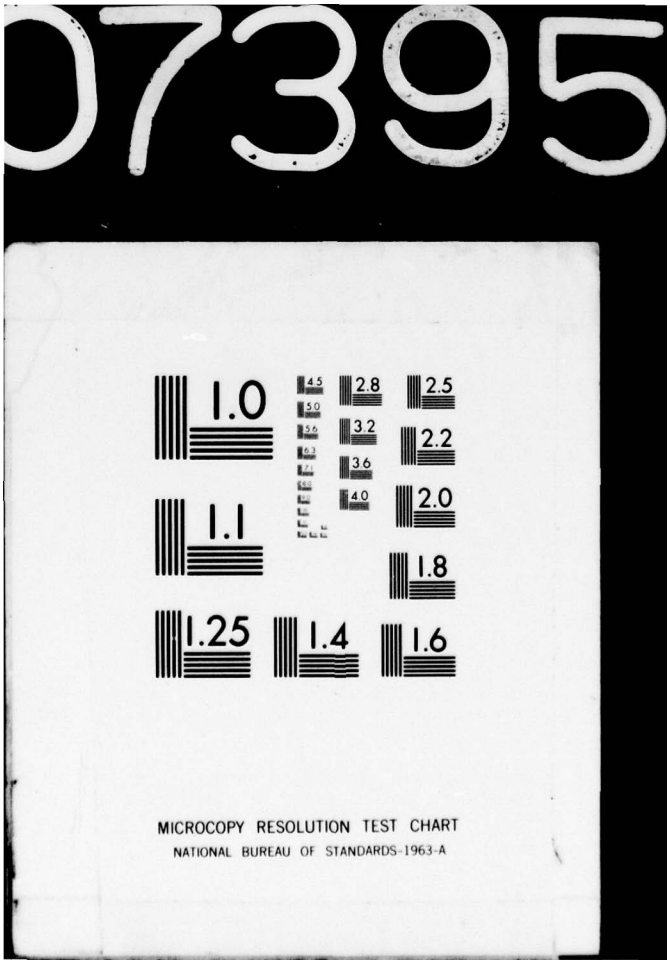
4. Areas and Pointers

To compensate for the lack of a heap, we extended our original design, adding

AD-A073 958 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 15.(U)
JUL 79 M L DERTOUZOS N00014-75-C-0661
UNCLASSIFIED LCS-PR-15 NL

2 OF 2
AD A073958

END
DATE FILMED
10-79
DDC


07395
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

areas and pointers. An area is a block of storage set aside for dynamic allocation. The entire block may be allocated in a stack frame, in a manner similar to declaring a large array. Objects may be created in an area at will (so long as the storage set aside is not exhausted). Areas are similar to the collections of Euclid [21], but may contain objects of differing sizes and types. Objects allocated in an area are referred to by using pointers. A pointer is restricted to refer either to nothing (a null pointer) or to objects of one particular type in one particular area. Note that pointers do not refer to variables, in keeping with our object-oriented view.

An object allocated in an area may be modified (by operations of its type), but cannot be assigned to or explicitly destroyed. Instead, an area can be garbage collected, using an implementation provided by the programmer. The area mechanism is designed to permit a wide variety of implementations, allowing the implementer freedom to adjust time-space efficiency trade-offs.

Because every pointer into an area, and indeed every variable that might directly or indirectly contain a pointer into an area, must contain the name of the area in its type, when the scope of an area is exited there can be no dangling references into the area. Hence the storage associated with the area may be safely reclaimed. Enforcement of the above rule is done by the normal type-checking function of the compiler.

5. Conclusion

We have designed a language that supports abstract data types without requiring a heap. This involved adjusting the basic semantics of CLU to allow stack implementation, and then solving some problems that derived from this change. It is interesting to note that features similar to those described above have appeared in designs having the same goal. For example, Alphard [22] has a mechanism very similar to our selectors, and the DOD/1 specification [20] leads to mechanisms similar to our size parameters.

The resulting language is definitely more complex than CLU. Furthermore, it appears that the added complexity is inherent, since Alphard and the DOD/1 designs have the same sorts of complex features. The only reason for pursuing designs such as ours is the efficiency gained by omitting heap management and garbage collection. As more efficient garbage collection methods become available, which recent developments in parallel and incremental garbage collections indicate may happen in the near future, languages such as CLU will have less efficiency penalty. Hence we hope that our design will become obsolete and simpler languages will be acceptably efficient for almost all purposes.

F. SYNTHESIS OF SYNCHRONIZATION CODE

When dealing with abstract data objects that are shared among different concurrent processes, some form of control over the ordering of accesses to objects is required. M. Laventhal has developed a method for automatically synthesizing source language synchronization code, given a synchronization constraint expressed in a problem specification language [23]. The following sections describe the specification language and the synthesis method.

1. The Specification Language

The data objects with which this research is concerned are the sort provided in programming languages supporting the notion of abstract data types, such as CLU [1], ALPHARD [15], or Simula [24]. In these languages, associated with an abstract data type there is a set of basic procedures, or operations, and only these operations are allowed to manipulate the lower-level representation of the abstract objects. Higher-level procedures can access the objects only by invoking the operations.

A basic assumption is that the units upon which synchronization should be performed are the basic operations of the abstract data type. Only these operations are allowed to access and manipulate the data representation of the abstract objects, and so it is here that decisions can be made as to what pattern of accesses is necessary to maintain internal consistency. The centralization of these operations in one module (such as a CLU cluster) permits a single expression of constraints to cover all accesses of the objects. Since the programming language ensures that all accesses are made through the basic operations, the discipline required for synchronization can be enforced universally; this would not necessarily be true if higher-level procedures were chosen for synchronization. On the other hand, to the user of an abstraction the exact implementation of the basic operations is unknown (and may change without warning). Synchronization constraints at any lower level, i.e., involving code internal to these operations, therefore would not be meaningful to the user. It is exactly at the level of the basic operations of a data type that the two viewpoints of the implementer and the user can and should be resolved in a smooth interface. This is true for the synchronization component of the interface just as much as for the data component.

A strict division is assumed between the synchronization and data manipulation functions involved in accessing a shared data object. This is based on the philosophy that the task of synchronization belongs in a separate language construct, whose sole function is synchronization. The operations of the abstract data type do not contain synchronization code, but are written assuming synchronization exists that is sufficient to prevent any conflicts between concurrent operation activations. Synchronization is taken to be uniform across all objects of the same type, reflecting the belief that a type consists not only of data manipulation operations but their synchronization as well. That is, all objects of a given type are synchronized in the same way.

The model of synchronization used assumes there is an abstract protection mechanism that conceptually surrounds each data object on which accesses must be synchronized (see Figure 4). This mechanism, called the guardian of the data abstraction, monitors all manipulations of the object, in a manner similar to the "secretary" concept proposed in [25]. Through this monitoring, the guardian is able to maintain the synchronization state of the resource, an abstract representation of the history of accesses to the object. (This is to be contrasted with the "data state" of the object, which is the state explicitly manipulated by the basic operations.) The guardian uses the synchronization state information to temporarily block any process attempting an access that is unsafe in the current synchronization state. The blocked process is allowed to proceed when the synchronization state has changed in such a way that the access can safely occur.

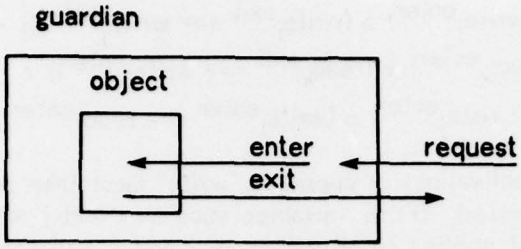


Figure 4. The Guardian Model.

Accessing an abstract data object consists of invoking one of the operations of the type to which the object belongs. The distinguishing features of the approach concern the structure imposed on synchronized accesses of the object. As indicated in Figure 4, every access involves a fixed sequence of events. The process wishing to make an access first communicates this desire to the guardian; this is called the "request" event for the access. When the guardian permits the initiation of the access on the actual data object, the "enter" event occurs. The termination of the access is communicated to the guardian in the "exit" event.

The guardian model assumes that the set of all events concerning a particular data object is totally ordered. That is to say, while many procedure activations can be executing concurrently, only one request, enter, or exit event associated with a given object can occur at a time. This total ordering property is comparable to the fact that the "arrival ordering" for any particular actor in [26] is total, and relies ultimately on some sort of arbitration mechanism for each data object.

The guardian model paradigm of request-enter-procedure body execution-exit forms the basis of the specification language. A synchronization specification is written for an abstract data type, and is intended to apply independently to every object of that type. The specification expresses a constraint on the ordering of accesses to an object, and represents the only such constraint. This means that any ordering of events consistent with the specification is valid, and in particular that procedure activations are allowed to execute in parallel unless constrained otherwise by the specification.

Specifications for synchronization problems can be written in a language based on this model. An access of an object is denoted by the operation being performed and the activation number of the access using that operation within the history of accesses of the object. For example, p_i represents the i-th activation of operation p on an object. One of the events associated with this access is denoted by adding the event name as a superscript, e.g., $p_i^{request}$ being the "request" for the p_i access. The total ordering of all events associated with a given data object is defined by a relation denoted ==>.

Specifications in the language are written as predicate calculus formulas that constrain the time ordering relation ==>. For example, a specification for a readers-writers database with priority given to writers [27] is expressed as:

$$((write_i^{enter} \Longrightarrow write_j^{enter}) \supset (write_i^{exit} \Longrightarrow write_j^{enter})) \wedge$$
$$((write_i^{exit} \Longrightarrow read_k^{enter}) \vee (read_k^{exit} \Longrightarrow write_i^{enter})) \wedge$$
$$((write_i^{request} \Longrightarrow read_j^{enter}) \supset (write_i^{enter} \Longrightarrow read_j^{enter}))$$

The first clause states that activations of operation "write" must take place one at a time and in the order requested. (Free variables such as i and j are universally quantified, so this constraint applies to all values of i and j, and therefore to all activations of "write.") The second clause requires activations of operations "write" and "read" to be mutually exclusive, in that one must exit before the other can enter. The third clause gives priority to activations of "write" over those of "read," by stating that any activation of "write" that is requested before an activation of "read" has actually entered must enter first.

2. Synthesis of the Solution Specification

The problem specification is a non-procedural representation of a synchronization property at the level of events. In synthesizing an implementation for a specified property, it is necessary to derive a procedural representation of the same property. The synthesis is accomplished in two steps. The first stage is a transformation from non-procedural to procedural form. The intermediate form is called the solution specification. It can be described without reference to the exact details of particular source language constructs. The second stage constructs an actual implementation.

In the solution specification, the form chosen to represent the synchronization state of an object is the number of events of each class that have occurred in the history of the object. This quantity is denoted by the term "count(ec)," where ec is the class of events to which the quantity refers. An event class exists for each event (request, enter, exit) for each operation. For example, $count(p^{request})$ represents the total number of "request" events for operation p.

Of the three types of events, "request" and "exit" events are generated from outside the synchronization mechanism, while "enter" events are generated by the synchronization mechanism itself (see Figure 4 above). For this reason, "enter" events are the only ones whose timing can be controlled by the synchronization code. The abstract solution to a synchronization problem can be represented by the condition on the synchronization state that must be true for each kind of "enter" event to be allowed.

Applying the method to the example specification above for the readers-writers database with writers' priority, the following conditions are derived for the "enter" event classes. For $read^{enter}$:

$$count(write^{request}) = count(write^{exit})$$

For $write^{enter}$:

$$count(write^{enter}) = count(write^{exit}) \wedge count(read^{enter}) = count(read^{exit})$$

The condition under which an activation of "read" may enter is that the number of

"request" and "exit" events for operation "write" be equal, so that there are no unfulfilled requests or active executions of "write." The condition under which an activation of "write" may enter is that (1) the number of "enter" and "exit" events for operation "write" be equal, so that there are no other active executions of "write," and (2) the number of "enter" and "exit" events for operation "read" be equal, so that there are no active executions of "read" as well.

A synchronization state expressed in terms of the number of events of each class lacks sufficient power to represent solutions to many synchronization properties of interest. For this reason, solution specification conditions must sometimes refer not only to the current state but also to synchronization states associated with previous events in the computation. When a condition derived for the solution specification involving the current state is insufficient, the synthesis algorithm uses previous state information to correct the condition. It is also possible to specify properties in which the synchronization behavior depends on the arguments to the procedure activations that constitute the accesses of interest. This is reflected by the presence in the solution specification of argument-dependent conditions.

Besides serving as a convenient intermediate form for the synthesis algorithm, the solution specification also can be used to test the soundness of the original problem specification. In particular, a potential for deadlock or starvation within a synchronization constraint can be determined by testing the conditions under which different kinds of accesses are blocked.

3. Synthesis of the Implementation

Once the solution specification is derived, it is fairly straightforward to implement it in terms of a suitable source language synchronization mechanism. Each quantity of the form count(ec) must be represented by an integer variable. This variable is initialized to 0, and incremented by 1 at the appropriate point in the access. Before the "enter" event may occur, the corresponding condition from the solution specification must be tested. If the condition is not true, then the process must wait until it becomes satisfied. To protect the integrity of the synchronization data, the incrementing and testing must be done within critical sections of code that are guaranteed to be indivisible. A convenient construct with which to implement the synchronization code is the monitor [28], since the procedures of a monitor are implemented as critical sections and the monitor "wait" and "signal" mechanisms are suitable for controlling the blocking and unblocking of processes.

The monitor for a data type contains three procedures for each operation p of the type. These procedures represent the three event classes associated with p, and are named p_request, p_enter, and p_exit. The form that operation p must take is illustrated in Figure 5. The identifier "m" is the name of the constructed monitor, and v is the vector of arguments to operation p.

```
p = proc ... ;
      call m.p_request(v);
      call m.p_enter(v);
        .
        . (body of p)
        .
      call m.p_exit(v);
end p;
```

$p^{request}$: increment count($p^{request}$) by 1
p^{enter}: wait until entry condition is satisfied,
then increment count(p^{enter}) by 1
execute body of operation p
p^{exit}: increment count(p^{exit}) by 1

Figure 5. Monitor calls within operation p.

Figure 6 shows the monitor derived for the writer's priority specification shown above. In this monitor, the integer variables wr, wn, wx, rn, and rx, represent count($write^{request}$), count($write^{enter}$), count($write^{exit}$), count($read^{enter}$), and count($read^{exit}$), respectively. There are also condition variables writeentry and readentry, corresponding to the conditions in the solution specification; their associated Boolean predicates are

readentry: $wr = wx$
writeentry: $wn = wx \wedge rn = rx$

Notice that count($read^{request}$) does not appear in the solution specification, so that no variable is needed for it, and thus a procedure read_request is not required.

The choose statement used in Figure 6 is a variation of Dijkstra's guarded command [29]. The meaning of this statement is the following: The "guards" B_i are simply Boolean expressions. If one or more of these guards is true, then one of the true guards B_j is selected (non-determinately, but the choice must be fair) and the corresponding statement s_j is executed. If none of the guards is true, then the statement terminates.

4. Evaluation

The specification language has proved to be quite convenient for writing synchronization specifications. Since all of the standard logical operators of predicate calculus can be used, and formulas of arbitrary complexity constructed, any constraint on time ordering can be expressed. The specifications are relatively easy to write and to understand, since each logical operator has a natural interpretation. The extensibility of the language permits a complex specification involving many constraints to be expressed as a conjunction of individual clauses, each one specifying a single constraint. This feature, illustrated in [23] enhances both constructability and comprehensibility.

```
wpdb = monitor;
        wr, wn, wx, rn, rx: integer;
        readentry, writeentry: condition;

        write_request = procedure;
                wr := wr + 1;
                choose
                        condition$queue(readentry) ∧ wr = wx:
                                condition$signal(readentry);
                        condition$queue(writeentry) ∧ wn = wx ∧ rn = rx:
                                condition$signal(writeentry);
                end;
        end write_request;

        write_enter = procedure;
                if wn ≠ wx ∨ rn ≠ rx then condition$wait(writeentry) end;
                wn := wn + 1;
                choose
                        condition$queue(readentry) ∧ wr = wx:
                                condition$signal(readentry);
                        condition$queue(writeentry) ∧ wn = wx ∧ rn = rx:
                                condition$signal(writeentry);
                end;
        end write_enter;

        write_exit = procedure;
                wx := wx + 1;
                choose
                        condition$queue(readentry) ∧ wr = wx:
                                condition$signal(readentry);
                        condition$queue(writeentry) ∧ wn = wx ∧ rn = rx:
                                condition$signal(writeentry);
                end;
        end write_exit;

        read_enter = procedure;
                if wr ≠ wn ∨ wn ≠ wx then condition$wait(readentry); end;
                rn := rn + 1;
                choose
                        condition$queue(readentry) ∧ wr = wx:
                                condition$signal(readentry);
                        condition$queue(writeentry) ∧ wn = wx ∧ rn = rx:
                                condition$signal(writeentry);
                end;
        end read_enter;
```

```
        read_exit = procedure;
                rx := rx + 1;
                choose
                        condition$queue(readentry) ∧ wr = wx:
                                condition$signal(readentry);
                        condition$queue(writeentry) ∧ wn = wx ∧ rn = rx:
                                condition$signal(writeentry);
                end;
        end read_exit;

        wr, wn, wx, rn, rx := 0, 0, 0, 0, 0;
end wpdb;
```

Figure 6. Monitor for writers' priority database.

The efficiency of synthesized implementations is reasonable for a large class of problems, assuming fairly simple code optimization techniques are employed. The fact that all synchronization code manipulates only integer-valued quantities, and that entry conditions always consist of linear equalities or inequalities of such quantities, keeps the implementations efficient. The efficiency can be enhanced if obvious optimizations are applied to the results of the straightforward synthesis. For example, by a simple analysis one can prove that, in the monitor in Figure 6, the choose statements in write_request, write_enter, and read_enter can all be eliminated, as can the first clause of the choose statement in read_exit.

There are some limitations in the synthesis method. Due to the relatively rigid structure of the solution specification, certain interesting synchronization properties cannot be captured. For example, the commonly used first-come-first-served specification cannot be expressed in the solution specification. Further, the monitor implementation may be extremely complex and inefficient for certain classes of specifications, such as those that depend on procedure arguments whose range of values is unknown; it is not known, however, if such specifications are really useful in practice.

Another serious problem with the synthesis method is its practicality. The algorithm as it stands can be used manually by a person to implement a synchronization constraint expressed in the specification language, or to informally check a hand-coded implementation. However, further work is needed to automate the algorithm. The synthesis method described here can only be viewed as a starting point for pursuing this general approach.

REFERENCES

1. Liskov, Barbara H.; Snyder, L. Alan; Atkinson, Russell R.; and Schaffert, J. Craig. "Abstraction Mechanisms in CLU." Communications of the ACM, Vol. 20 No. 8 (August 1977), 564-576.

2. Liskov, Barbara H., and Snyder, L. Alan. Structured Exception Handling. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 155-1. Cambridge, Ma., September 1978.

3. Liskov, Barbara H.; Moss, J. Eliot; Schaffert, J. Craig; Scheifler, Robert W.; and Snyder, L. Alan. CLU Reference Manual. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 161. Cambridge, Ma., July 1978.

4. Schaffert, J. Craig. A Formal Definition of CLU. M.I.T., Laboratory for Computer Science, LCS/TR-193. Cambridge, Ma., January 1978.

5. Scheifler, Robert W. A Denotational Semantics of CLU. M.I.T., Laboratory for Computer Science, LCS/TR-201. Cambridge, Ma., June 1978.

6. "Programming Methodology Group." Progress Report July 1976 - July 1977. M.I.T., Laboratory for Computer Science, LCS/PR-XIV. Cambridge, Ma., 135-161.

7. Gries, David, and Gehani, N. "Some Ideas on Data Types in High-Level Languages." Communications of the ACM, Vol. 20 No. 6 (June 1977), 414-420.

8. Principato, Robert N., Jr. A Formalization of the State Machine Specification. M.I.T., Laboratory for Computer Science, LCS/TR-202. Cambridge, Ma., July 1978.

9. Parnas, David L. "A Technique for the Specification of Software Modules, With Examples." Communications of the ACM, Vol. 15 No. 5 (May 1972), 330-336.

10. Price, W. L. "Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems." Ph.D Thesis, Carnegie-Mellon University, Pittsburgh, Pa., June 1973.

11. Robinson, Lawrence, and Levitt, Karl. "Proof Techniques for Hierarchically Structured Programs." Communications of the ACM, Vol. 20 No. 4 (April 1977), 271-283.

12. Roubine, O., and Robinson, Lawrence. SPECIAL Reference Manual. Stanford Research Institute, Technical Report, CSG-45. Stanford, Ca., August 1976.

13. Berzins, Valdis A. Abstract Model Specifications for Data Abstractions. Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, forthcoming.

14. Hoare, C. A. R. "Proofs of Correctness of Data Representations." Acta Informatica, Vol. 1 No. 4 (1972), 271-281.

15. Wulf, William A.; London, Ralph; and Shaw, Mary. "An Introduction to the Construction and Verification of Alphard Programs." IEEE Transactions on Software Engineering, Vol. SE-2 No. 4 (December 1976), 253-265.

16. Kapur, Deepak. Towards a Theory of Data Abstractions. Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, forthcoming.

17. Goguen, Joseph A.; Thatcher, James W.; Wagner, Eric G.; and Wright, Jesse B. "Abstract Data Types as Initial Algebras and the Correctness of Data Representations." Proceedings of Conference on Computer Graphics, Pattern Recognition and Data Structures, IEEE, Los Angeles, Ca., 1975, 89-93.

18. Guttag, John V. The Specification and Application to Programming of Abstract Data Types. University of Toronto, Computer Systems Research Group, CSRG-59. Toronto, Canada, 1975.

19. Moss, J. Eliot. Abstract Data Types in Stack Based Languages. M.I.T., Laboratory for Computer Science, LCS/TR-190. Cambridge, Ma., February 1978.

20. Steelman. Department of Defense, Requirments for High Order Computer Programming Languages, June 1978.

21. Lampson, Butler W.; Horning, James J.; London, Ralph L.; Mitchell, James G.; and Popek, Gerald L. "Report on the Programming Language Euclid." SIGPLAN Notices, Vol. 12 No. 2 (February 1977).

22. Wulf, William A., et. al. An Informal Definition of Alphard (Preliminary). Carnegie-Mellon University, Computer Science Department, Report CMU-CS-78-105. Pittsburgh, Pa., February 1978.

23. Laventhal, Mark S. Synthesis of Synchronization Code for Data Abstractions. M.I.T., Laboratory for Computer Science, LCS/TR-203. Cambridge, Ma., July 1978.

24. Dahl, Ole-Johan, and Hoare, C. A. R. "Hierarchical Program Structures." Structured Programming. New York: Academic Press, 1972, 175-220.

25. Dijkstra, Edsger W. "Hierarchical Ordering of Sequential Processes." Operating Systems Techniques. Edited by C. A. R. Hoare and R. Perrott. New York: Academic Press, 1972, 72-93.

26. Hewitt, Carl; Bishop, Peter; and Steiger, Richard. "A Universal Modular Actor Formalism for Artificial Intelligence." Proceedings of Third International Joint Conference on Artifical Intelligence, Stanford, Ca., 1973, 235-245.

27. Courtois, P. J.; Heymans, F.; and Parnas, David L. "Concurrent Control with 'Readers' and 'Writers'." Communications of the ACM, Vol. 14 No. 10 (October 1971), 667-668.

28. Hoare, C. A. R. "Monitors: An Operating System Structuring Concept." Communications of the ACM, Vol. 17 No. 10 (October 1974), 549-557.

29. Dijkstra, Edsger W. "Guarded Commands, Nondeterminacy and Formal Derivation of Programs." Communications of the ACM, Vol. 18 No. 8 (August 1975), 453-457.

PROGRAMMING METHODOLOGY GROUP 116

Publications

1. Berzins, Valdis, and Kapur, Deepak. Denotational and Axiomatic Definitions for Path Expressions. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 153-1. Cambridge, Ma., November 1977.

2. Greif, Irene G. "A Language for Formal Problem Specification." Communications of the ACM, Vol. 20 No. 12 (December 1977), 931-935.

3. Liskov, Barbara H. "Practical Benefits of Research in Programming Methodology." AFIPS Conference Proceedings, Vol. 47, Anaheim, Ca., June 1978, 666-667.

4. Liskov, Barbara H., and Jones, Anita K. "A Language Extension for Expressing Constraints on Data Access." Communications of the ACM, Vol. 21 No. 5 (May 1978), 358-367.

5. Liskov, Barbara H., and Snyder, L. Alan. Structured Exception Handling. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 155. Cambridge, Ma., December 1977.

6. Liskov, Barbara H.; Snyder, L. Alan; Atkinson, Russell R; and Schaffert J. Craig. "Abstraction Mechanisms in CLU." Communications of the ACM, Vol. 20 No. 8 (August 1977), 564-576.

7. Moss, J. Eliot. Abstract Data Types in Stack Based Languages. M.I.T., Laboratory for Computer Science, LCS/TR-190. Cambridge, Ma., February 1978.

8. Schaffert, J. Craig. A Formal Definition of CLU. M.I.T., Laboratory for Computer Science, LCS/TR-193. Cambridge, Ma., January 1978.

9. Scheifler, Robert W. "An Analysis of Inline Substitution for a Structured Programming Language." Communication of the ACM, Vol. 20 No. 9 (September 1977), 647-654.

10. Scheifler, Robert W. A Denotational Semantics of CLU. M.I.T., Laboratory for Computer Science, LCS/TR-201. Cambridge, Ma., June 1978.

11. Scheifler, Robert W., and Snyder, L. Alan. CLU Information Package. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 154. Cambridge, Ma., November 1977.

Accepted for Publication

1. Liskov, Barbara H. "Remarks on the Construction of Large Programs." To be published in The Impact of Research on Software Technology. Edited by P. Wegner. Cambridge, Ma.: M.I.T. Press.

2. Liskov, Barbara H., and Berzins, Valdis. "An Appraisal of Program Specifications." To be published in The Impact of Research on Software Technology. Edited by P. Wegner. Cambridge, Ma.: M.I.T. Press.

Theses Completed

1. Garrard, Stephen C. "An Arithmetic Compiler for the Digital Acoustic Signal Simulator (DASS)." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

2. Laventhal, Mark S. Synthesis of Synchronization Code for Data Abstractions. Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1978.

3. Moss, Eliot. Abstract Data Types in Stack Based Languages. S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, November 1977.

4. Principato, Robert N., Jr. A Formalization of the State Machine Specification Technique. S.M. and E.E. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

5. Schaffert, J. Craig. A Formal Definition of CLU. S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, January 1978.

6. Scheifler, Robert. A Denotational Semantics of CLU. S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

7. Shienbrood, Eric R. "A Translator for the Language CLUMAC." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

Theses in Progress

1. Berzins, Valdis. "Abstract Model Specification for Data Abstractions." Ph.D. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, January 1979.

2. Bloom, Toby. "An Analysis of Synchronization Methods for Modular Programs." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, January 1979.

3. Kapur, Deepak. "Towards a Theory of Data Abstractions." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1979.

4. Leach, Paul. "Designing a Garbage Collector in a Strongly Typed Language." S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, January 1979.

5. Snyder, L. Alan. "A Structured, Verifiable Machine Architecture to Support an Object-Oriented Language." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, January 1979.

6. Zilles, Stephen N. "Data Algebra: A Specification Technique for Data Structures." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1979.

Talks

1. Liskov, Barbara H. Chairman of Panel Session on Program Specification Techniques, IFIP Congress, Toronto, Canada, August 1977.

2. Liskov, Barbara H. "An Appraisal of Program Specifications." Conference on Research Directions in Software Technology, Providence, R. I., October 1977.

3. Liskov, Barbara H. "CLU Abstraction Mechanisms and Their Implementation." SIGPLAN Meeting, Boston, Ma., November 1977; Cornell University, Ithaca, N. Y., December 1977.

4. Liskov, Barbara H. "Abstraction Mechanisms in CLU." University of Waterloo, Waterloo, Canada, November 1977; Bell Laboratory, Indian Hill, Il., November 1977.

5. Liskov, Barbara H. Session Chairman, History of Programming Languages Conference, Los Angeles, Ca., June 1978.

PROGRAMMING TECHNOLOGY

Academic Staff

A. Vezza, Group Leader

J. C. R. Licklider

Research Staff

E. R. Banks
J. M. Berez
M. S. Blank
M. F. Brescia
M. S. Broos
S. W. Galley
D. S. Gerson
P. D. Lebling
C. L. Reeve
D. Sherry

Graduate Students

T. A. Anderson

Undergraduate Students

S. H. Berez
B. T. Berkowitz
N. M. Butt
D. L. Dill
T. K. Johnson
G. E. Kaiser
P. C. Lim
S. C. Phillips
T. J. Platt
B. J. Roberts
W. A. Seltzer
S. H. Soto
W. W. St. Clair

Support Staff

S. P. Briggs

PROGRAMMING TECHNOLOGY

A. INTRODUCTION

The Programming Technology group is engaged in two distinct research and development programs. The program in Morse code has as its main goals the development of the conceptual insight necessary to develop a computerized Morse code operator and the design and implementation of a prototype of such a computer system (COMCO-I)[1]. The Morse code program covers four areas: signal processing, Morse code transcription, sender recognition, and understanding of the network conversations among operators that are carried on in a special language consisting of "Q-signs," "Pro-signs," and "Call-signs." The other research program is concerned with the facilitation of interpersonal communication through the use of computer message systems[1]. The work on interpersonal communication has involved the design and implementation of a computer message system that embodies in it a model, as yet very simple, of an organization. The model is used to track action status and to aid the communication process.

B. MORSE CODE

COMCO-I, the prototype computerized Morse code operator, is composed of three major subsystems.

1. A signal acquisition and processing module produces a file of mark (dot and dash) and space durations based on analysis of a signal.

2. A transcription module converts the mark and space durations into a lattice of possible transcriptions of the message, where each branch of the lattice is a vocabulary element from a large but finite vocabulary. This module begins by performing a MAUDE-like [1] assignment of each code element to its apparent type, and then passes that result to COMDEC (COmputerized Morse DECoder), which builds the lattice of transcriptions.

3. Finally, the transcriptions suggested by COMDEC are evaluated by CATNIP, a parser based on augmented transition networks, which attempts to derive coherent, "grammatical" transmissions from them.

1. Domain Models

It is clear that good Morse operators have conceptual models of the Morse code environment that they use to help them perform their task. They have models of Morse "sounds"--sequences of dots and dashes with rhythm and timing information--and map these sounds into the letters and words. They have models of the language constructs that are used, be they English, another natural language, or the chatter language. Operators form models of other operators' mannerisms and use these models in the translation and understanding processes and in identifying other operators. Operators also have models of the Morse code and radio domains. It is common knowledge that "TH" is often sent with a short space between the letters, so that a machine often interprets it as "6" (thus "6E" is really "THE"). Similarly, "AN" is often interpreted as "P" (thus "PD" is really "AND"). In the radio domain, knowledge about

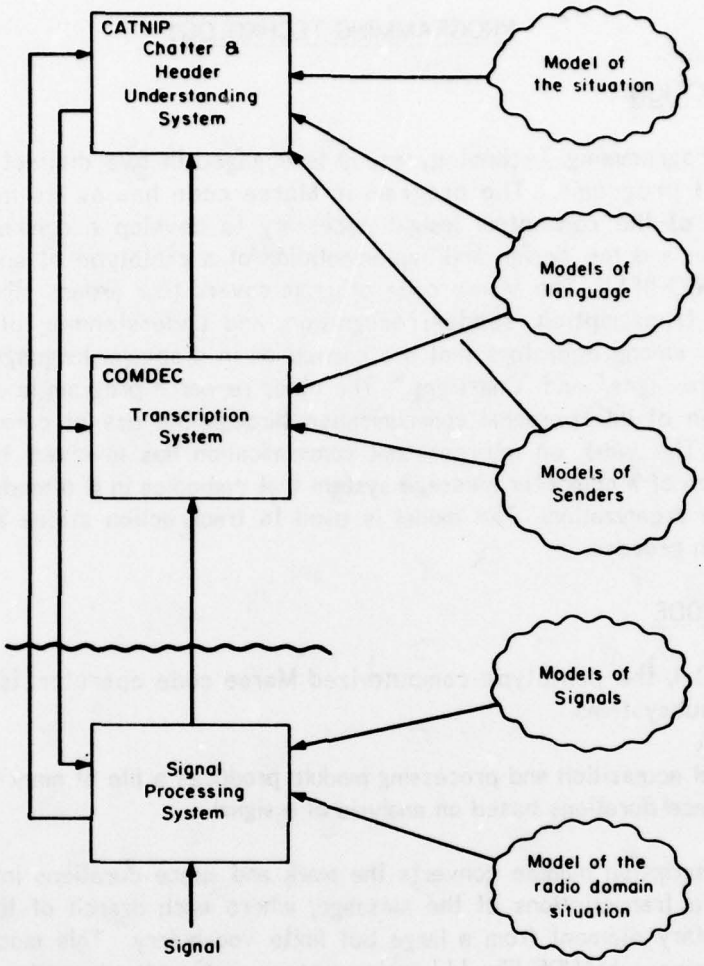


Figure 1. The Three Major Modules of the Morse Code System and the Domain Models They Use.

the spectral frequency to which relevant operators' transmitters are tuned, what a particular transmitter sounds like, how a signal fades and returns--all these form the models that help operators identify, track, transcribe and understand one another. It is the human being's ability to interpret Morse sounds in the context of such mental models that allows her or him to perform so well.

At this point a slight digression is in order. Listening to a Morse code conversation among a group of operators, one notices three distinct aspects of the conversation. These correspond to (i) network chatter, (ii) message headers, and (iii) message bodies. The chatter section is often very poorly sent. Characteristically, many letters and words are slurred or separated and corrupted by other operator lapses. Yet receiving operators have little difficulty understanding chatter, because they have a model of the global situation: what question was asked by whom, who is currently waiting on the network, who has message traffic for whom, and so forth. This model and the ability to understand the conversation is vitally important to translation.

Headers of messages are structured but, unfortunately, not rigidly. Again, to translate them correctly one must have some understanding of what headers are about. For instance, dates may be sent as "8 Dec 78" or "8 12 78" or "81278" and times may be sent as "1000Z" or "1000". We have written the dates and times in an ideal manner, but, in fact, they might be--as a result of operator lapses--segmented quite differently, so that parts of numbers are run together or a number is split apart, and one or more numbers might contain a mark-error. One other aspect of numbers is very important. All numbers in Morse code are five marks long--see [2] under Morse code--yet they are often abbreviated and sent as what are called cut-numbers. Again, context is often required to perform translation correctly.

The body of amateur radio message traffic is typically English with some abbreviations. To attempt to understand all of the English language would be far beyond the scope of this research. We have built into the system just enough knowledge to let it perform in a creditable fashion. Our experience indicates that a vocabulary, some rules about where numerals can occur in text, rules about how to handle error signs, and a measure of closeness in a Hamming-like space (for correcting operator induced irregularities) are absolutely essential to the correct translation of plain text. Our experience suggests that knowledge about idiosyncratic and irregular behavior of individual operators facilitates translation.

Military message traffic differs slightly because the body may consist either of plain text or of cipher groups. A message sent in cipher has no language context, but knowledge about the number of groups in a message, the number of characters in a group, and whether groups are alphabetic, numeric, or mixed is necessary to translate such a message.

Figure 1 shows a block diagram of the three major modules of the Morse code system COMCO-I. Also shown are the necessary domain models required by each module in order for it to perform its task properly. The wavy line in the diagram indicates that the signal processing system, which is composed of special hardware and a PDP-11 computer, is not integrated with the other major modules which are COMDEC, the transcription (or translation) module, and CATNIP, the chatter and header understanding module. The last two are software modules written in MDL (a LISP-like language) [3] and running under TOPS-20 [4] and ITS [5]. Experiments are conducted independently for the signal processing system, and human intervention is required to transfer the results to the other two modules. COMDEC and CATNIP are well integrated, with appropriate feedback, and externally they appear to behave as one system.

A few phrases about each domain model may prove helpful:

a. Model of the radio domain situation--how the individual transmitters of interest sound, i.e., whether a transmitter has any characteristic envelope or carrier distortions, and if so what kind and a measure of the amounts.

b. Model of the Network situation--which operators are logged into the network, which are off control frequency, which are on control frequency and where each operator's transmitter is tuned relative to those of the other operators on his frequency. This last bit of information turns out to be quite important, as we

will show later, even though all the operators are working in a thirty to fifty Hertz band.

c. Models of senders--the irregularities a particular sender may introduce, such as his or her idiosyncrasies of language, a proclivity to introduce extraneous dots or omit dots, etc.

d. Models of language--the full gamut of possibilities are required for parsing and understanding chatter, but we have rather simple models for handling message bodies such as a vocabulary and some simple rules for handling some special constructs and numbers.

e. Models of the situation--the system must know when a question is asked and the possible range of expected answers; it must know that a frequency change has been ordered or negotiated and how to respond appropriately; and so forth.

2. Morse Code Transcription

The capabilities of COMDEC, the Morse code transcription module, were expanded and improved during the past year (Lebling, Sherry). The major thrust of development during the year has been to augment COMDEC's abilities in transcribing Morse code into plain text sent in an environment more closely approximating conditions of live communication between operators. There were two areas in which this effort concentrated, each of which will be discussed in turn:

a. Improving the transcriber's performance on transmissions composed primarily of "network chatter," the specialized language of the Pro-signs, Q-signs and Call-signs used by Morse code operators;

b. Design and implementation of an interface between the COMDEC transcriber and CATNIP, a parser for Morse traffic network interactions.

2.1 Network Chatter

The part of Morse code known familiarly as "network chatter" has several characteristics which set it apart from transmissions in English. Specifically, because the vocabulary is limited (typically under 1000 common words, Pro-signs, Q-signs and Call-signs), and the context of the transmission is often rigidly prescribed, the quality of the code sent is considerably lower than for English transmissions. The code is likely to contain many more space-errors and mark-errors than a message in plain English.

The most common type of error in "network chatter" is that of running words together. There are two main reasons why this is true. First, certain sequences are perceived by the sender as single words, such as a Q-sign followed by a question mark. Second, certain sequences, through repetition, have become so easy to send that the operator sends them mechanically, tending to compress them into one burst of code.

In "network chatter," the many numbers involved are often sent as "cut numbers." A cut number is a standard Morse code number in which the leading or trailing dash sequence has been replaced by one (usually longer than average) dash. For example, a cut "0" (normally five dashes) consists of a single long dash, often indistinguishable from a Morse "T". A cut "1" looks like an "A", and a "9" like an "N". To correctly transcribe these transmissions, cut numbers must be correctly distinguished from the letters they resemble.

Another common problem in "chatter" is that, in certain contexts, numbers are so common that the receiving operator expects them, and can compensate for a high level of errors in sending. It is common for a sender to transmit single cut numbers if the context demands a number (this is rare in English transmissions), or to transmit a "5" (normally five dots, and the only digit composed only of dots) as any non-zero number of dots.

COMDEC has been modified to deal with these and other problems that have arisen in the transcription of "chatter." Some of the specific modifications that have been made include the following.

1. The heuristics have been improved that decide when a letter-space or mark-space is a possible word-space. Specifically, a simple routine to recognize punctuation marks and other commonly run-together code elements was written. Additionally, certain code sequences commonly run together because of their "rhythm" (such as dot-dash-dot-dash) are now recognized.

2. A module was added to the standard transcription sequence which recognizes and transcribes sequences of V's, which are commonly sent as an aid in receiver tuning.

3. The number-transcription module has been expanded to deal with cut numbers on the same basis as standard five-mark numbers.

4. Words are now checked as they are placed in the transcription lattice to see if they introduce a "number context." If they do, the number-transcription routine is called and informed that such a context exists (which makes it more tolerant of errors and cut numbers).

5. The format in which code samples are stored has been expanded to allow miscellaneous information about the code--most importantly, the locations of sender changes--to be stored with the code sample.

These, and other changes not mentioned specifically, have improved COMDEC's performance on "chatter" considerably, as will be seen later.

2.2 CATNIP Interface

The second area of COMDEC development (Lebling, Sherry) has been the design and implementation of an interface between COMDEC and CATNIP (see below). This development included a redesign of the top-level transcriber of COMDEC. Previously, COMDEC transcribed an entire "message," which could include several sender changes,

as one unit. It now treats a transmission by a sender as the basic unit, running all transcription modules on that transmission before even running MAUDE on the next transmission.

The interface with CATNIP was designed to allow the two to pass information back and forth without requiring each to know the data structures of the other, or to view the other as anything more than a "black box." The interface module is thus very simple. CATNIP calls it with a pointer into the code, and COMDEC returns the transcriptions at that point in the lattice, sorted by quality (best transcription first). With each transcription is a simple evaluation of it ("good," "indifferent," "bad") and a pointer to where the next transcription comes from if this one is correct.

What goes on "behind CATNIP's back" is that the interface module has a record of which parts of the message have been transcribed, and, if necessary, it transcribes more of the message before returning to CATNIP. If the area in which CATNIP is working has already been transcribed, the information it wants is already there. Consequently, COMDEC and CATNIP operate more or less in parallel and, given the simplicity of their interface, could even be in different processes.

2.3 An Example

Here is an example of a transcription of "chatter," in which two senders, code-named ROCK and SALT, are trying to establish contact. The special abbreviations used in this example are as follows:

@	"Previous portion of code is erroneous; ignore it."
ANS	answer
DE	"This is ..."
K	Over
PSE	please
R	Roger
QRK	"What is my intelligibility?" or "Your intelligibility is ..."
QRO	"Shall I increase transmitter power?" or "Increase transmitter power."
QRQ	"Shall I send faster?" or "Send faster."
QSA	"What is my signal strength?" or "Your signal strength is ..."
QSV	"Shall I send V's?" or "Send V's."
QTC	"How many messages do you have?" or "I have ... messages."
V	[aid in receiver tuning]

MAUDE's transcription: (Curly brackets {} surround a sequence of marks sent as one letter.)

SALT: {VVV} V{VV} {VVV} R OC K ROCK RO CK DE SAL T SAL T QSA? K

{VVV} {VVV} V{VV} ROCKROCKROCK DE SALT SALT QS{.-..--..} QRK? T A

{VVV} {VVV} {VVV}ROCK{.-.---}CKROCKROCKDE SI @ S AL T SAL T SAL T QSA? QRTA? QSANONO QRKNONTM QSV QSV K

{VVVVV}VVVIAIAV ROCK ROCKROCKROCKRMTCKROCKROCKROCKDE SALT QSA? QRK? QSANONOQRKNONOWEST ANS PSE ANS QRQ QRQ QTC QTC QTC K

ROCK: {VV}V SAL T DE ROCKQSA2 QSA2QRK@E QRK@ QS A? QR K? K

SALT: ROCKTIE SALTQSAH QRKH TTAAETTT ? K A •

ROCK: DE ROCKRRR QROQROQSVQSVK

SALT: DE SALT R {VV}VV{VVVVVV}{VVVVVV}V{VV}V E ...

COMDEC's transcription: (Curly brackets {} enclose an untranscribed mark. Pointed brackets <> enclose a word obtained by correcting a mark-error. Square brackets [] enclose an error sign and the error to be ignored. COMDEC's errors are underlined.)

SALT: VVV VVV VVV ROCK ROCK ROCK DE SALT SALT QSA ? K

VVV VVV VVV ROCK ROCK ROCK DE SALT SALT QSA ? QRK ? <M>

VVV VVV VVV ROCK ROCK ROCK ROCK DE [xxxxx @] SALT SALT SALT QSA ? QRK ? QSA NO NO QRK NO NO QSV QSV K

VVV ROCK ROCK ROCK ROCK ROCK ROCK ROCK ROCK DE SALT QSA ? QRK ? QSA NO NO QRK NO NO <PSE> ANS PSE ANS QRQ QRQ QTC QTC QTC K

ROCK: VVV SALT DE ROCK QSA 2 QSA 2 QRK 5 QRK [@] QSA ? QRK ? K

SALT: ROCK DE SALT QSA <5> QRK <5> QRO ? <K> {T}

ROCK: DE ROCK <NR> R QRO QRO QSV QSV K

SALT: DE SALT R VVV ...

COMDEC made four errors in selecting the best transcription, but CATNIP, with its superior "knowledge," was able to correct all of them. In each case the correct transcription was available in the lattice, but COMDEC did not select it as the best. CATNIP was able to reject the incorrect transcriptions and select the correct ones in each case: "K" instead of "<M>", "5" instead of "[@]", nothing instead of "{T}", and "R R" instead of "<NR>".

3. CATNIP Chatter and Header Understanding System

CATNIP is a semantic-syntactic augmented-transition-network (ATN) parser (Sherry, Kaiser, Vezza) that chooses a path through the lattice of possible translations created by COMDEC.

CATNIP uses ATN diagrams to choose the correct word from a lattice of possible translations. It starts in a certain state of the transition network, and progresses from one state to another, depending on the next word or words in the lattice. With each

state is associated a list of words, and with each word a new state. CATNIP matches the list of words from the state with the list of words possible at that point in the translation lattice; matches yield valid new states.

If that were all, the network would simply be an unaugmented transition network. However, CATNIP retains a context, which it changes (usually with every word) and which can be tested when it is trying to match the words. The context includes such things as who is the sender of the current transmission, who is the receiver, who is the net controller, and so on. ATNs, as opposed to unaugmented transition networks, are good for parsing grammars that are dependent on the context and on past occurrences [6].

Naturally, ambiguities creep in. Sometimes more than one match is possible; CATNIP allows for this by processing one of the new valid states and saving all the others. The context at that point is saved with the states that were saved. CATNIP has the ability to return to the saved states and try those alternate paths.

Finally, CATNIP also has a limited understanding of the events on the net. Understanding these events is important in understanding the state of the net at any point (how many operators are working, who they are, who is talking, etc.) and it is important in choosing the correct word at a particular point in the translation.

The context is used as the "understanding" part of CATNIP. Take the following transmission as an example:

ROCK ROCK ROCK DE SALT SALT QSA ? QRK ? K.

Upon completion of the parse, the parser would retain a context that contained the information that the receiver was ROCK, the sender was SALT, and SALT had asked ROCK two questions: "What is my signal strength?" and "What is my intelligibility?"

Retaining this kind of context helps find the right translation and decide later ambiguities (such as who is the receiver at a certain point, if he or she was not explicitly named). The successive contexts also furnish a synopsis of the entire session after the parser is finished.

CATNIP is a recursive procedure that allows one to name ATN diagrams of simple structures (such as Q-signs that are often used), and to use those as parts of other diagrams without actually duplicating the simple diagrams. Thus a more structured "grammar" can be created without over complicating the data base.

Figure 2 shows a typical ATN diagram. The circles with either an "S" or a number in them indicate the states of the transition network. A diagram is always entered in the start, "S", state and a return from a diagram is always achieved from one of the states allowing a return. A state allowing a return to a caller is indicated by partial shading of the circle. Each italicized ARC label such as Header indicates a call to another diagram; in lower case are labels such as "location", which indicate that the labeled input that will parse is a location (such as "BOSTON" or "BOS"); arcs labeled with a number sign "#" mean that a number is acceptable as input (with the parenthesized statement indicating the meaning of the number, e.g., (nr-gr) means

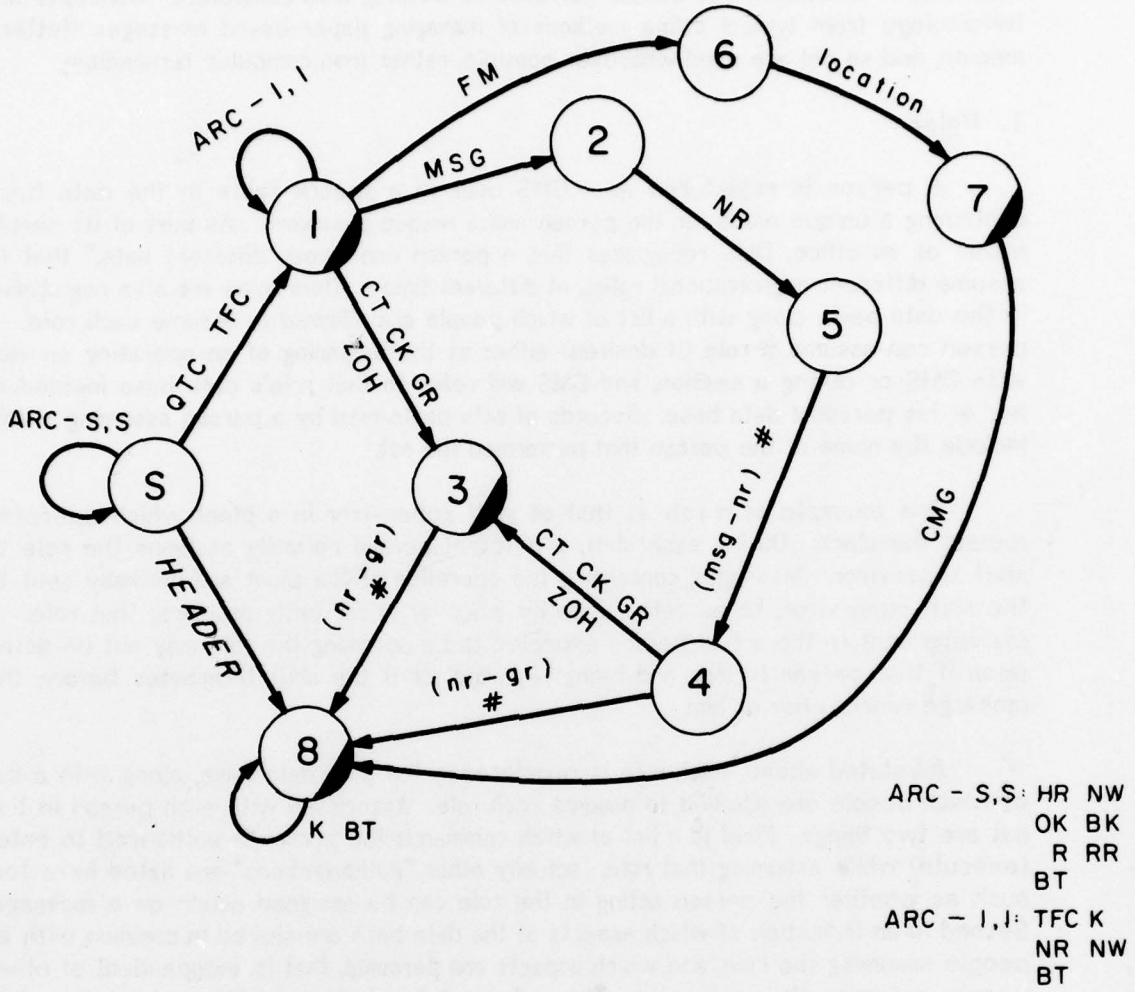


Figure 2. Augmented Transition Network Diagram Called "Traffic Header."

number of groups); and arcs labeled in upper case mean that literal input of one of the specified labels is acceptable. The system currently contains about 25 diagrams with an average complexity of the one shown in Figure 2.

C. INTERPERSONAL COMMUNICATION

The research in interpersonal communication has continued at a low level with further design and implementation of the Data-based Message Service (DMS) [1] as described below (Broos, Berez, Brescia, Galley, Vezza). DMS is "data-based" because the messages it manages are data in a number of similar on-line relational data bases, which may contain thousands or even tens or hundreds of thousands of messages.

The general model on which DMS is designed is that of a typical office. The interface at an intelligent terminal between DMS and a user is designed to be

comfortable and familiar to people not used to working with computers. Concepts and terminology from typical office methods of managing paper-based messages (letters, memos, and so on) are used wherever possible, rather than computer terminology.

1. Roles

A person is registered as a DMS user in a special table in the data base containing a unique name for the person and a unique password. As part of its simple model of an office, DMS recognizes that a person can "wear different hats," that is, assume different organizational roles, at different times. Thus roles are also registered in the data base, along with a list of which people are allowed to assume each role. A person can assume a role (if desired) either at the beginning of an operating session with DMS or during a session, and DMS will refer to that role's data base instead of her or his personal data base. Records of acts performed by a person assuming a role include the name of the person that performed the act.

One example of a role is that of shift supervisor in a plant which operates around the clock. During each shift, a different person normally assumes the role of shift supervisor. Messages concerning the operation of the plant are normally sent to the shift supervisor, to be acted upon by whoever is currently assuming that role. A message sent to the actual person expected to be assuming the role may not be acted upon if that person is sick and being replaced, or if the shift terminates before the message reaches her or him.

As stated above, each role is registered in the DMS data base, along with a list of which people are allowed to assume each role. Associated with each person in this list are two things. First is a list of which commands the person is authorized to enter (execute) while assuming that role; actually other "authorizations" are listed here too, such as whether the person acting in the role can be assigned action on a message. Second is an indication of which aspects of the data base are shared in common with all people assuming the role, and which aspects are personal, that is, independent of other people assuming the same role. These "aspects" include everything about the data base except the messages and data-base indexes themselves, which are always shared by everyone that assumes the role, to prevent confusion. (Since the "desk-top bins" and "file cabinet" are in effect mutually exclusive parts of an office's storage, they are always either all personal or all common.)

The utility of authorizing certain people, but not others, to perform certain commands while assuming a role should be clear; for example, an administrative assistant might be authorized to assume the supervisor's role to read and file all messages except those marked personal.

To see the utility of common versus personal aspects of a role's data base, consider the above examples. A shift-supervisor role could be registered with all aspects common to all users of that role, if that were the desired mode of operation, so that, for example, a message seen by one person would appear to have been seen by the other people assuming the role also. Thus, each person assuming the role has complete authority (and responsibility) for changing the role's data base in any way. On the other hand, when an administrative assistant uses DMS and assumes the supervisor's role, the role data might say that seeing a message would remove a "not-

seen" tag from the message only for the assistant and not for the supervisor, while sending a message would add a "sent" tag for both of them. Thus, the assistant may have authority to send messages (presumably routine ones) for the supervisor (as a role) but the system must be clever and not change the supervisor's "seen" tags when the assistant is acting the supervisor's role.

This role mechanism provides a useful way to construct a public "bulletin board." A bulletin board is a role that can be assumed by any person at all, but no variable aspects of the bulletin board data base are shared in common: each person can keep track, with the "text-not-seen" tag, of which messages she or he has not read; each person can note, with the "pending" bin, which messages he or she ought to take some action on; each person can display or print messages in personalized formats; each person can forward a message to any user to obtain a copy (really just a citation) in a personal data base; and so forth.

The notion of treating a bulletin board as a role is quite natural, though perhaps surprising when one first encounters it. In a typical paper-world office environment, one must go to a particular place (desk, work station, etc.) to assume a functional role. Similarly, one must go to a particular public place to scan and read messages posted on a public bulletin board. It is relatively difficult to act as a person or role while reading messages on a bulletin board, simply because one is not physically in the place where one has the facilities and tools for taking action. If one wants a copy of a bulletin board message, one has to use the nearby copying machine and then carry the copy back to one's work station. A DMS bulletin board is thus a close analog of one in the paper world, with the additional abilities to personalize it a great deal.

2. Other Changes

To supplement the many ways that DMS aids formal communication, both within an organization and between it and the outside world, there is now a way for a user to send transient messages, called alerts. An alert is not stored in the data base; it is just displayed on the recipient's terminal. There are two kinds of alerts: "must-see" alerts are guaranteed to be seen by the user, either immediately or at the beginning of his or her next session; "see-if-here" alerts are either seen immediately by the user (if he or she is currently using DMS) or thrown away. Alerts are also sent by DMS itself, to inform users of changes in the data base of which they should be aware. For example, when DMS delivers a new message to a user, it sends a "see-if-here" alert: if the user is currently using DMS, she or he may want to see the message immediately; otherwise, he or she will no doubt see the message during the next session anyway.

The top line of a DMS terminal's display is now used as a status line. (The top line was previously used to "flash" short responses to the user's commands. Command responses are now put right next to the displayed command on the screen.) On the status line, the "virtual terminal" module displays current status information, including the user's name and current role, operating-system load, the amount of central-computer processing time used so far, and the date and time.

A DMS folder (the analog of a manila file folder in a conventional office) now has a kind of audit trail stored with it: a "log" that simply lists all changes made to the

folder, namely adding and removing messages and changing the folder itself (for example, access to it). To complement this unstructured log, the "virtual terminal" module of DMS now has the capability to search the text currently displayed in the terminal's "information window," both visible text and that which is scrolled out of view; thus a user can find references to a given message in a folder log by first displaying the log on the terminal and then searching in the terminal for the message's identifying number. A DMS folder now can have annotations stored with it, in analogy to written annotations on the outside or inside of a manila folder; a user with any access at all to the folder can see all its annotations.

Fixed sequences of commands can now be "canned" and put in the data base by a privileged user, and thereafter other users can activate a command sequence and then copy the commands in order, one at a time, from the data base to the active (bottom) line of the terminal's "command window." Each command can in turn be entered as it is or modified in the terminal first. "Canned" command sequences are useful both for demonstrating or teaching the features of the message service and for entering frequently-used but (mostly) unchanging command sequences.

A search-cost estimator was added, which estimates how many messages must be examined (rather than found through indexes) in an imminent search and, if the number exceeds a threshold, requires the user to confirm the need for the lengthy search before proceeding. The estimate is the best one available for the implementation that is both conservative and not itself time-consuming.

Soon after the new TOPS-20 system (see below) was installed, DMS was easily converted to run under it as well as under the Tenex operating system. A standard script of commands ran about three times as fast as on a PDP-10 KA processor.

D. OTHER PROJECTS

1. New Mainframe

In January 1978 the Laboratory took delivery of a DECsystem-2050T. Design work began on modifications to the TOPS-20 operating system to support device-independent display-terminal use by user programs (Gerson).

The hardware configuration of the new system includes:

a. 512 x 1024 x 36 bits of core storage (two MB20 units)

b. 2 x 1024 x 36 bits of semiconductor cache storage

c. 120 million x 36 bits of disk storage (three RP06 units on one RH20 channel), divided by the operating system into two units of "public structure" and one unit of demountable structure

d. two nine-track tape drives (TU45 units on one RH20 channel)

e. 32 terminal lines (two DH11 units)

f. an ARPA network interface, but with no available connection to the network as yet (one is on order).

2. Keyword Extraction and Document Classification

Work continued this year at a relatively low level on the keyword extractor and the document classifier which is based on it (Dill) [1]. Most of the effort was devoted to testing the document classifier in order to determine its possible usefulness for several applications, and to gain insight on the most desirable course its development should follow in the future. As a consequence, the document classifier is in the process of being extensively modified at this time.

The ability of the document classifier to identify the topic area of an English paragraph would suggest at least two obvious general applications: the automatic classification of documents (such as the preparation of a newspaper index), and the detection and identification of the mention of a given topic area or areas. The current emphasis is on the latter application.

The document classifier is automatic, and it takes advantage of information provided by EPARSE (an English parser) [1] and the keyword extractor. EPARSE parses an English sentence, providing syntactic information such as the parts of speech of the words in the sentence and their functions, in addition to morphological information and information stored directly in the dictionary. It also provides limited semantic information both by specifying the position of the various words in a hierarchy of all the words in its dictionary--in which objects are specified as parts or types of other objects (the "kind" relation)--and by specifying a context for many nouns, verbs, and adjectives. A context is the "topic of discussion" in which a word is likely to occur (for example, "calculus" would probably be mentioned in the context MATHEMATICS). From this information, the keyword extractor selects what it believes to be the most useful keywords in the document.

The document classifier attempts to match these keywords against words which have been previously determined to represent certain categories. The information about categories is contained in a structure called a model. When a keyword in any of the categories matches a keyword from a document, the weight (a real number) immediately following it is added to a cumulative total which is used as an indicator of how well that particular document fits the model of categories in which the particular keyword appears. If, when the keywords are exhausted, the total is greater than a model's threshold, the document is classified with the name of the model and the accumulated weight.

A typical model looks like this:

Name of model: TAXATION
Threshold: 1.7

Classifiers: "tax credit" (1.0), "gasoline tax" (1.0), "luxury tax" (1.0), "automobile tax" (1.0), "import duty" (1.0), "tax administration" (1.0), "excise tax" (1.0), "social security" (0.6), "value-added tax" (1.0), "property tax" (1.0), "sales tax" (1.0), "income tax" (1.0)

Key nouns: "tax" (0.8), "crop" (0.8)
Key-noun meanings: TAX (1.0)
Key proper names: "internal revenue service" (1.0)
Key verbs: "tax" (0.6)
Key-verb meanings: [none]
Verb-object combinations: "deduct tax" (1.0), "evade tax" (1.0), "pay tax" (1.0), "collect tax" (1.0)
Subject-verb combinations: [none]
Transformations: [none]
Generalizations: [none]
Contexts: PUBLIC-ADMINISTRATION (0.2), ECONOMICS (0.2), GOVERNMENT (0.2), MONEY (0.2)
Unknowns: "revenue" (0.4), "treasury" (0.4)

Testing of the keyword extractor and document classifier against extracts from newspapers has indicated that the most useful information for classification is the limited semantic information provided by the parser, particularly the contextual information. This is not particularly surprising, since we want to classify most of the documents on the basis of their topic areas, which are exactly the contexts in which they are likely to occur. This information cannot be provided in any detail without some sort of syntactic analysis, because it depends on the parser's ability to provide syntactic constraints on word meanings in the sentence. Thus, the contextual information is derived from entire sentences and paragraphs; not from single words. For example, a document on the subject of calculus would probably be in the topic area of MATHEMATICS, and we might want to classify it accordingly. However, the word "calculus" also occurs in the domain of medicine, for an abnormal deposit, such as a kidney stone. In order to distinguish between these two meanings, we would probably have to use contextual information derived from other parts of the sentence, paragraph and article, which may be constrained by the syntax of the sentences in which they occur.

3. Experimental English Parser

A simple but powerful parser for a restricted subset of the English language for use as the human interface in a restricted domain was developed (Anderson, Blank, Lebling). The parser handles nouns, verbs, direct objects, indirect objects, adjectives and incomplete specification, the last by responding with a question. The parser was tested in a game situation called "Dungeon." The capability of the parser to handle a larger domain such as that of the DMS world of office automation was also investigated (Anderson, Broos, Lebling).

The Dungeon world consists mainly of objects (nouns) and actions (verbs). The parser handles mainly imperative sentences, plus a few simple interrogatives ("What is a grue?"). The relationship between nouns and verbs in the parser is divided between the verbs and the objects being acted upon. For example, in Dungeon, the user's sentence "Give bomb to thief" is evaluated by allowing the "thief" (an object) to have the first crack at the parsed sentence. The thief's "give" component (a function or "handler") checks to see if a bomb is being given and, if so, refuses to accept it, printing an appropriate message and terminating the evaluation. The "troll," however, is not so smart. If one attempted to give the bomb to the troll, which has no "give"

handler, the "bomb" would be given a chance to handle the sentence. If the bomb had no "give" handler, or if its "give" handler saw nothing interesting about the sentence, the sentence would eventually be handled by the global "give" handler by default.

A parallel example exists in office management scenarios. If a shipping clerk told the system to "Ship fuel on United #564", it would recognize "United #564" as a commercial air flight (a class of objects) and would further recognize that it is a passenger flight as opposed to freight. The passenger flight handler could examine the object of the sentence "fuel," see that one of its attributes was "dangerous," tell the clerk that dangerous cargo cannot be shipped on commercial passenger flights, and terminate the evaluation. If it were a freight flight, there would be no reason to check the attributes of the cargo, so the indirect object handler would let the parse continue. The direct object, "fuel", would get next crack at the sentence. It may look at the indirect object and test its own attributes to see if the pressure and temperature changes encountered in air freight render that mode of transportation unsuitable.

At first glance, parcelling the decisions out to so many different units of the model appears to be a mistake, inviting confusion. However, it makes a lot of sense to localize the decisions in the units most directly affected. In the above example, the air freight handler doesn't need to know anything about fuel except that it is dangerous. It has its own small set of rules, one of which is that you can't carry dangerous cargo on a passenger flight. Presumably, the rules were defined by some expert in the air freight business, who knows nothing about the effects of temperature and pressure changes on different types of fuel. Such checks are properly made by the fuel module, whose rules are defined by a fuel expert. If the evaluation managed to get by both modules, it would be handled by the "ship" function, which would simply check to see if the types of the object and indirect object were legal for the action being performed, i.e., it would make sure that the direct object was a physical object and that the indirect object was a mode of transportation. Checks about the availability of the flight in question and whether the proposed cargo will fit would properly be the province of the air freight module.

Incomplete instructions would also be handled by the different modules. "Ship fuel" would elicit the response "By what means of transportation" from the "ship" handler, which would notice the absence of any indirect object. If the user had been "talking" about fuel, however, "Ship on United #564" could easily supply the direct object. It would of course, have to inform the user of what assumptions it made, such as "Fuel shipped on United airlines freight flight #564, departing National airport at 12:02 p.m." Similarly, "Ship fuel by air" might cause the "air" handler to query the user as to what airline, what flight, etc., or it might be able to schedule the shipment on the first available flight with enough space to handle it, if it had access to that type of information.

A mechanism such as this would provide a way to make the system "smarter" incrementally. For example, the "air" handler mentioned above could be implemented in the simple way at first, and upgraded later when the air freight reservation data became available.

One of the extensions of the parser which would have to be made would be to enable it to understand and handle more than one object with the same name.

Currently, there is only one thief, one troll, etc., although there are objects of the same type that are distinguished by color. The parser would have to be able to handle similar objects as an aggregate ("ships"), as the subset of an aggregate ("container ships"), as a dynamic subset ("container ships on the East coast with cargoes of machine parts"), and as individuals ("The Mara Maru"). Some actions may be applied only to individuals of a class, some only to aggregates, and some to either. The extended parser would have to understand set operators. Also, the mechanisms for resolving "it" would have to be extended to "them."

The illusion of English understanding created by the current parser in the game Dungeon is due in large part to the fact that the user is unknowingly using a very restricted set of nouns and verbs. The restricted nature of this set is not normally apparent to the user because the nature of the game itself dictates what operations and objects are appropriate. Thus users restrict their own choices naturally. Whether this could be carried over into an office situation where the range of actions and objects is finite, but much larger than the range in the Dungeon, is still an open question.

4. Recognition of Cursive Script

A computer recognition procedure can be defined as one in which the input is some representation of an object and the output is another representation of the same object. In the case of computer recognition of cursive script, the input consists of the cursive stroke(s) used to represent some word, and the output is the coded character-string representation of the same word.

Cursive-script recognizers can be separated into two categories: character-oriented and word-oriented. A character-oriented recognizer attempts to separate those parts of a script stroke that correspond to the individual letters in the word. The cursive representation of each segment identified as a character is then replaced with its coded character representation.

There are two difficulties with character-oriented systems. The first is that the probability of correctly recognizing a script sample goes down exponentially as the length of the word represented by the script goes up. For a word of length N, the probability of correctly recognizing it is the Nth power of the probability of recognizing a single character. Character-oriented recognizers also have a disadvantage in that they can produce a character string that is not a legal word [8,9].

Word-oriented recognizers operate by translating the cursive script of an entire word into the character representation of an entire word. This is done by determining general attributes of the script sample, such as the number of loops. The recognizer then uses the attributes that have been determined from a sample to refer to a data base. This data base contains the information about attributes for all the words the recognizer is expected to recognize. Only the attributes for legal words are stored in the data base, so only legal words can result, no matter what the script sample looks like. The disadvantage of this type of recognizer is that a mistake is global to the word and not localized to a character. Thus the recognizer in making a mistake could produce a word which in no obvious way resembles the intended word [7,10].

In an undergraduate thesis this year, Platt reported on a word-oriented recognizing system called SCRAP (SCRipt Attribute Processor)[11]. SCRAP was designed to produce more than one result for each sample, with information on the degree of fit of the sample to the attributes in the data base. Some of the operating characteristics of the SCRAP program were determined by experimentation using over a thousand samples of words recorded from nine subjects.

SCRAP takes as input the coordinate data from a script sample and produces as output a string which is the ASCII representation of a word corresponding to the unknown sample. Actually SCRAP produces a list of strings, and thus SCRAP could be useful as a front-end processor for a more advanced system, such as a text-editing or interpersonal message system (see above). This list of strings is ordered in terms of how well each word matches the unknown sample as a possible result, with the first words being the better matches. If, because of the writer's style, the sample does not fit ideal script form, then SCRAP may not produce the intended word as the best possible match, but the intended word may be one of the other strings produced. This would allow a program that can use additional information, such as context, to process the list produced by SCRAP. Such a system could produce a more reliable recognition of a sample than could a recognizer alone, either a word- or a character-oriented recognizer.

The data base used by SCRAP is formed with data taken from real samples of script. Thus another function of SCRAP is to store information concerning attributes as well as to retrieve them. In addition SCRAP can be used to describe the distribution of word objects in the data space imposed by the partitioning of the data space by the attributes being used. The seven attributes used by SCRAP are the number of strokes, the number of ascending characters, the number of descending characters, the number of local vertical maxima, the total amount of curvature, the total amount of positive (counter-clockwise) curvature, and the number of times the curvature changes sign.

5. An Urdu/English Text Editor

An undergraduate thesis [12] by Butt describes the design and implementation of a bi-lingual text entry and editing system for Urdu and English. It is an interactive character-oriented system and uses variable width characters displayed on a raster-scan bit-map display monitor. The graphic representations used for the Urdu characters are standardized so as to bring out the similarities between the different forms of each character. The system is implemented in LISP and, in addition to the standard editing features, allows mixing English and Urdu.

Urdu is one of the most widely used languages in the Indian subcontinent. Besides being the second official language of the people of Pakistan, Urdu is also used in many parts of India.

Urdu is similar in script to Arabic and Persian. It is written from right to left and the characters have a number of different graphical representations although there are only thirty seven (37) characters in the alphabet. The graphic symbol used for a character depends on the position of the character in the word and its right and left neighbors. The symbols are standardized and would not appear exactly as they would

in cursive script. When writing Urdu by hand, one has a greater degree of freedom and as a result the script is highly embellished.

The keyboard presents only 37 characters to the user and the appropriate form of the character is automatically chosen by the editor. Furthermore the editor chooses proper alternate forms as necessary when characters are inserted or deleted during an edit operation. The different graphical representations of the characters have different widths, in order to make the words readable, and the editor takes the variable widths into account to display the text in its proper connected form.

REFERENCES

Note: The form XXX.nn.nn denotes a Programming Technology Group document.

1. M.I.T. Laboratory for Computer Science. Progress Report XIV. Cambridge, Ma. 1978.

2. Webster's New Collegiate Dictionary. Springfield, Ma.: G. & C. Merriam Company, 1974.

3. Galley, S.W. and Pfister, Greg. MDL Primer and Manual. Cambridge, Ma.: M.I.T. Laboratory for Computer Science, 1977.

4. DECSYSTEM-20 User's Guide. Maynard, Ma.: Digital Equipment Corporation, 1978.

5. Eastlake, D.; Greenblatt, J.; Holloway, J.; Knight, T.; and Nelson, S. ITS 1.5 Reference Manual. Cambridge, Ma.: M.I.T. Laboratory for Computer Science, 1969.

6. Woods, William A. "Transition Network Grammars for Natural Language Analysis." Communications of the Association for Computing Machinery, Vol 13 No 10, (1970).

7. Kolers, Paul A. and Eden, Murray, eds. Recognizing Patterns. Cambridge, Ma.: M.I.T. Press, 1968.

8. Sayre, Kenneth M. Machine Recognition of Handwritten Words; A Project Report. Notre Dame, In.: Philosophic Institute for Artificial Intelligence, University of Notre Dame, 1973.

9. Frishkopf and Harmon, "Machine Reading of Cursive Script." Information Theory. Edited by Colin Cherry. Washington D.C.: Butterworths, 1961.

10. Earnest, L.D. "Machine Recognition of Cursive Writing." Information Processing. 1962.

11. Platt, Timothy J. "A Preprocessor for a Script Recognition System." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

12. Butt, Nayyar. "A Bi-lingual Text Entry and Editing System for Urdu/English." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

Publications

1. Licklider, J.C.R. "Future Directions in Computer Networking Applications." Computer Networking in the University: Success and Potential, Proceedings EDUCOM Fall Conference 1976. Chapter 4, 27-39. Princeton, N.J.: Interuniversity Communications Council, 1977.

2. Licklider, J.C.R. "Library Network: Should They Deal with Containers or Contents of Knowledge?" Computer Networking in the University: Success and Potential, Proceedings EDUCOM Fall Conference 1976. Chapter 15, 113-117. Princeton, N.J.: Interuniversity Communications Council, 1977.

3. Louis T. Rader et al. Review of a New Data Management System for the Social Security Administration. Panel on Social Security Administration Data Management System, Committee on Telecommunications--Computer Applications, Assembly of Engineering, National Research Council, National Academy of Sciences, Washington D.C., 1978.

4. Louis T. Rader et al. Review of Requirements Definition and Systems Architecture of a New Data Management System for the Social Security Administration. Panel on Social Security Administration Data Management System, Committee on Telecommunications--Computer Applications, Assembly of Engineering, National Research Council, National Academy of Sciences, Washington D.C., 1978.

Theses Completed

1. Butt, Nayyar. "A Bi-lingual Text Entry and Editing System for Urdu/English." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

2. Platt, Timothy J. "A Preprocessor for a Script Recognition System." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1978.

Talks

1. Vezza, Albert. "Computers as a Communication Tool for Office and Home." M.I.T. Alumni Summer College, Cambridge, Ma. July 10-16, 1977.

2. Licklider, J.C.R. "Human Factors in Message Systems." Session on Electronic Mail I: Design. IFIP Congress 77. Toronto, Ontario, Canada. August 11, 1977.

3. Vezza, Albert. "Design of an Electronic Message System." Session on Electronic Mail I: Message System Designers. IFIP Congress 77. Toronto, Ontario, Canada. August 11, 1977.

4. Licklider, J.C.R. "Communication Between Systems Scientists and Human Factors Engineers." NATO Meeting. Brussels, Belgium. October 25-27, 1977.

5. Licklider, J.C.R. "Libraries and Information Networks." Conference on Library Systems. Pittsburgh, Pennsylvania. November 15, 1977.

6. Licklider, J.C.R. "Televistas Revisited: Technology-Based Opportunities for Public Television." Carnegie Commission on the Future of Public Broadcasting. Columbia, S.C. March 16, 1978.

7. Vezza, Albert. "Electronic Message Systems." Industrial Liaison Symposium on Office Automation, M.I.T., Cambridge, Ma. May 1978.

TECHNICAL SERVICES

Research Staff

K. T. Pogran, Group Leader

Undergraduate Students

C. Schieck

Support Staff

P. Baskin
O. Feingold
J. D. Ricchio

TECHNICAL SERVICES

The Technical Services group was formed on January 1, 1978 to fill a variety of Laboratory wide needs that have arisen as a result of our growing computational resources and expected future activities. The functions performed by the group include:

1. Completion and maintenance of hardware for the LCS Network.

2. Development and maintenance of standards for intra-Laboratory communications, terminals and related equipment.

3. Liaison for the Laboratory's ARPANET IMP and TIP.

4. Partial maintenance of LCS computers, terminals, and other peripherals; in particular, maintenance of the Laboratory's collection of approximately 75 Digital Equipment Corporation VT-52 CRT terminals.

5. Development, construction within the Laboratory, and sub-contracting to outside organizations of special equipment, e.g. interfaces, as required by new equipment acquisitions and LCS research group needs.

6. Development and maintenance of an LCS electronics laboratory facility for the construction and maintenance of equipment outlined above.

From January through July of 1978, the bulk of the work of the group centered around the development and debugging of the Local Network Interface (LNI) for the LCS Network. This work is described in the "Local Area Network Working Group" section of this report.

LABORATORY FOR COMPUTER SCIENCE PUBLICATIONS

TECHNICAL MEMORANDA

TM-10 Jackson, James N.
Interactive Design Coordination for the Building Industry
June 1970
AD 708-400

*TM-11 Ward, Philip W.
Description and Flow Chart of the PDP-7/9 Communications Package
July 1970
AD 711-379

*TM-12 Graham, Robert M.
File Management and Related Topics
(Formerly Programming Linguistics Group Memo No. 6, June 12, 1970)
September 1970
AD 712-068

*TM-13 Graham, Robert M.
Use of High Level Languages for Systems Programming
(Formerly Programming Linguistics Group Memo No. 2, November 20, 1969)
September 1970
AD 711-965

*TM-14 Vogt, Carla M.
Suspension of Processes in a Multi-processing Computer System
(Based on M.S. Thesis, EE Dept., February 1970)
September 1970
AD 713-989

*TM-15 Zilles, Stephen N.
An Expansion of the Data Structuring Capabilities of PAL
(Based on M.S. Thesis, EE Dept., June 1970)
October 1970
AD 720-761

TMs 1-9 were never issued.

*TM-24 Goldstein, Robert C., and Alois J. Strnad
The MacAIMS Data Management System
April 1971
AD 721-620

TM-25 Goldstein, Robert C.
Helping People Think
April 1971
AD 721-998

TM-26 Iazeolla, Giuseppe G.
Modeling and Decomposition of Information Systems for Performance Evaluation
June 1971
AD 733-965

*TM-27 Bagchi, Amitava
Economy of Descriptions and Minimal Indices
January 1972
AD 736-960

TM-28 Wong, Richard
Construction Heuristics for Geometry and a Vector Algebra Representation of Geometry
June 1972
AD 743-487

*TM-29 Hossley, Robert and Charles Rackoff
The Emptiness Problem for Automata on Infinite Trees
Spring 1972
AD 747-250

*TM-30 McCray, William A.
SIM360: A S/360 Simulator
(Based on B.S. Thesis, ME Dept., May 1972)
October 1972
AD 749-365

TM-31 Bonneau, Richard J.
A Class of Finite Computation Structures Supporting the Fast Fourier Transform
March 1973
AD 757-787

TM-32 Moll, Robert
An Operator Embedding Theorem for Complexity Classes of Recursive Functions
May 1973
AD 759-999

*TM-33 Ferrante, Jeanne and Charles Rackoff
A Decision Procedure for the First Order Theory of Real Addition with Order
May 1973
AD 760-000

*TM-34 Bonneau, Richard J.
Polynomial Exponentiation: The Fast Fourier Transform Revisited
June 1973
PB 221-742

TM-35 Bonneau, Richard J.
An Interactive Implementation of the Todd-Coxeter Algorithm
December 1973
AD 770-565

TM-36 Geiger, Steven P.
A User's Guide to the Macro Control Language
December 1973
AD 771-435

*TM-37 Schoenhage, A.
Real-Time Simulation of Multidimensional Turing Machines by Storage Modification Machines
December 1973
PB 226-103/AS

*TM-38 Meyer, Albert R.
Weak Monadic Second Order Theory of Succesor is not Elementary-Recursive
December 1973
PB 226-514/AS

TM-39 Meyer, Albert R.
Discrete Computation: Theory and Open Problems
January 1974
PB 226-836/AS

TM-40 Paterson, Michael S., Michael J. Fischer and Albert R. Meyer
An Improved Overlap Argument for On-Line Multiplication
January 1974
AD 773-137

TM-41 Fischer, Michael J., and Michael S. Paterson
String-Matching and Other Products
January 1974
AD 773-138

*TM-42 Rackoff, Charles
On the Complexity of the Theories of Weak Direct Products
January 1974
PB 228-459/AS

TM-43 Fischer, Michael J., and Michael O. Rabin
Super-Exponential Complexity of Presburger Arithmetic
February 1974
AD 775-004

TM-44 Pless, Vera
Symmetry Codes and their Invariant Subcodes
May 1974
AD 780-243

*TM-45 Fischer, Michael J., and Larry J. Stockmeyer
Fast On-Line Integer Multiplication
May 1974
AD 779-889

*TM-46 Kedem, Zvi M.
Combining Dimensionality and Rate of Growth Arguments for Establishing Lower Bounds on the Number of Multiplications
June 1974
PB 232-969/AS

TM-47 Pless, Vera
Mathematical Foundations of Flip-Flops
June 1974
AD 780-901

TM-48 Kedem, Zvi M.
The Reduction Method for Establishing Lower Bounds on the Number of Additions
June 1974
PB 233-538/AS

TM-49 Pless, Vera
Complete Classification of (24,12) and (22,11) Self-Dual Codes
June 1974
AD 781-335

TM-50 Benedict, G. Gordon
An Enciphering Module for Multics
B.S. Thesis, EE Dept.
July 1974
AD 782-658

*TM-51 Aiello, Jack M.
An Investigation of Current Language Support for the Data Requirements of Structured Programming
M.S. & E.E. Theses, EE Dept.
September 1974
PB 236-815/AS

TM-52 Lind, John C.
Computing in Logarithmic Space
September 1974
PB 236-167/AS

TM-53 Bengelloun, Safwan A.
MDC-Programmer: A Muddle-to Datalanguage Translator for Information Retrieval
B.S. Thesis, EE Dept.
October 1974
AD 786-754

*TM-54 Meyer, Albert. R.
The Inherent Computation Complexity of Theories of Ordered Sets: A Brief Survey
October 1974
PB 237-200/AS

TM-55 Hsieh, Wen N., Larry H. Harper and John E. Savage
A Class of Boolean Functions with Linear Combinatorial Complexity
October 1974
PB 237-206/AS

TM-56 Gorry, G. Anthony
Research on Expert Systems
December 1974

TM-57 Levin, Michael
On Bateson's Logical Levels of Learning
February 1975

TM-58 Qualitz, Joseph E.
Decidability of Equivalence for a Class of Data Flow Schemas
March 1975
PB 237-033/AS

*TM-59 Hack, Michel
Decision Problems for Petri Nets and Vector Addition Systems
March 1975
PB 231-916/AS

TM-60 Weiss, Randell B.
CAMAC: Group Manipulation System
March 1975
PB 240-495/AS

TM-61 Dennis, Jack B.
First Version of a Data Flow Procedure Language
May 1975

TM-62 Patil, Suhas S.
An Asynchronous Logic Array
May 1975

TM-63 Pless, Vera
Encryption Schemes for Computer Confidentiality
May 1975
AD A010-217

*TM-64 Weiss, Randell B.
Finding Isomorph Classes for Combinatorial Structures
M.S. Thesis, EE Dept.
June 1975

TM-65 Fischer, Michael J.
The Complexity Negation-Limited Networks - A Brief Survey
June 1975

*TM-66 Leung, Clement
Formal Properties of Well-Formed Data Flow Schemas
B.S., M.S. & E.E. Theses, EE Dept.
June 1975

*TM-67 Cardoza, Edward E.
Computational Complexity of the Word Problem for Commutative Semigroups
M.S. Thesis, EE & CS Dept.
October 1975

TM-68 Weng, Kung-Song
Stream-Oriented Computation in Recursive Data Flow Schemas
M.S. Thesis, EE & CS Dept.
October 1975

*TM-69 Bayer, Paul J.
Improved Bounds on the Costs of Optimal and Balanced Binary Search Trees
M.S. Thesis, EE & CS Dept.
November 1975

TM-70 Ruth, Gregory R.
Automatic Design of Data Processing Systems
February 1976
AD A023-451

*TM-71 Rivest, Ronald
On the Worst-Case of Behavior of String-Searching Algorithms
April 1976

*TM-72 Ruth, Gregory R.
Protosystem I: An Automatic Programming System Prototype
July 1976
AD A026-912

TM-73 Rivest, Ronald
Optimal Arrangement of Keys in a Hash Table
July 1976

TM-74 Malvania, Nikhil
The Design of a Modular Laboratory for Control Robotics
M.S. Thesis, EE & CS Dept.
September 1976
AD A030-418

TM-75 Yao, Andrew C., and Ronald I. Rivest
K+1 Heads are Better than K
September 1976

AD A030-008

*TM-76 Bloniaz, Peter A., Michael J. Fischer and Albert R. Meyer
A Note on the Average Time to Compute Transitive Closures
September 1976

TM-77 Mok, Aloysius K.
Task Scheduling in the Control Robotics Environment
M.S. Thesis, EE & CS Dept.
September 1976

AD A030-402

*TM-78 Benjamin, Arthur J.
Improving Information Storage Reliability
Using a Data Network
M.S. Thesis, EE & CS Dept.
October 1976

AD A033-394

TM-79 Brown, Gretchen P.
A System to Process Dialogue: A Progress Report
October 1976

AD A033-276

TM-80 Even, Shimon
The Max Flow Algorithm of Dinic and Karzanov:
An Exposition
December 1976

TM-81 Gifford, David K.
Hardware Estimation of a Process' Primary
Memory Requirements
B.S. Thesis, EE & CS Dept.
January 1977

TM-82 Rivest, Ronald L., Adi Shamir and Len Adleman
A Method for Obtaining Digital Signatures and
Public-Key Cryptosystems
(formerly On Digital Signatures and Public-Key Cryptosystems)
April 1977

AD A039-036

*TM-83 Baratz, Alan E.
Construction and Analysis of Network Flow Problem which Forces Karzanov Algorithm to $O(n^3)$ Running Time
April 1977

*TM-84 Rivest, Ronald L., and Vaughan R. Pratt
The Mutual Exclusion Problem for Unreliable Processes
April 1977

*TM-85 Shamir, Adi
Finding Minimum Cutsets in Reducible Graphs
June 1977
AD A040-698

TM-86 Szolovits, Peter, Lowell B. Hawkinson and William A. Martin
An Overview of OWL, A Language for Knowledge Representation
June 1977
AD A041-372

TM-87 Clark, David., editor
Ancillary Reports: Kernel Design Project
June 1977

TM-88 Lloyd, Errol L.
On Triangulations of a Set of Points in the Plane
M.S. Thesis, EE & CS Dept.
July 1977

TM-89 Rodriguez, Humberto Jr.
Measuring User Characteristics on the Multics System
B.S. Thesis, EE & CS Dept.
August 1977

TM-90 d'Oliveira, Cecilia R.
An Analysis of Computer Decentralization
B.S. Thesis, EE & CS Dept.
October 1977
AD A045-526

TM-91 Shamir, Adi
Factoring Numbers in O (log n) Arithmetic Steps
November 1977
AD A047-709

TM-92 Misunas, David P.
Report on the Workshop on Data Flow Computer and Program Organization
November 1977

TM-93 Amikura, Katsuhiko
A Logic Design for the Cell Block of a Data-Flow Processor
M.S. Thesis, EE & CS Dept.
December 1977

*TM-94 Berez, Joel M.
A Dynamic Debugging System for MDL
B.S. Thesis, EE & CS Dept.
January 1978
AD A050-191

TM-95 Harel, David
Characterizing Second Order Logic with First Order Quantifiers
February 1978

TM-96 Harel, David, Amir Pnueli and Jonathan Stavi
A Complete Axiomatic System for Proving Deductions about Recursive Programs
February 1978

*TM-97 Harel, David, Albert R. Meyer and Vaughan R. Pratt
Computability and Completeness in Logics of Programs
February 1978

TM-98 Harel, David and Vaughan R. Pratt
Nondeterminism in Logics of Programs
February 1978

TM-99 LaPaugh, Andrea S.
The Subgraph Homeomorphism Problem
M.S. Thesis, EE & CS Dept.
February 1978

TM-100 Misunas, David P.
A Computer Architecture for Data-Flow Computation
M.S. Thesis, EE & CS Dept.
March 1978
AD A052-538

TM-101 Martin, William A.
Descriptions and the Specialization of Concepts
March 1978

AD A052-773

TM-102 Abelson, Harold
Lower Bounds on Information Transfer
in Distributed Computations
April 1978

TM-103 Harel, David
Arithmetical Completeness in Logics of Programs
April 1978

TM-104 Jaffe, Jeffrey
The Use of Queues in the Parallel Data
Flow Evaluation of "If-Then-While" Programs
May 1978

TM-105 Masek, William J., and Michael S. Paterson
A Faster Algorithm Computing String
Edit Distances
May 1978

TECHNICAL REPORTS

*TR-1 Bobrow, Daniel G.
Natural Language Input for a Computer Problem Solving System,
Ph.D. Thesis, Math. Dept.
September 1964

AD 604-730

*TR-2 Raphael, Bertram
SIR: A Computer Program for Semantic Information Retrieval,
Ph.D. Thesis, Math. Dept.
June 1964

AD 608-499

*TR-3 Corbato, Fernando J.
System Requirements for Multiple-Access, Time-Shared Computers
May 1964

AD 608-501

*TR-4 Ross, Douglas T., and Clarence G. Feldman
Verbal and Graphical Language for the AED System: A Progress Report
May 1964

AD 604-678

*TR-6 Biggs, John M., and Robert D. Logcher
STRESS: A Problem-Oriented Language for Structural Engineering
May 1964

AD 604-679

*TR-7 Weizenbaum, Joseph
OPL-1: An Open Ended Programming System within CTSS
April 1964

AD 604-680

*TR-8 Greenberger, Martin
The OPS-1 Manual
May 1964

AD 604-681

TRs 5, 9, 10, 15 were never issued

*TR-11 Dennis, Jack B.
Program Structure in a Multi-Access Computer
May 1964
AD 608-500

*TR-12 Fano, Robert M.
The MAC System: A Progress Report
October 1964
AD 609-296

*TR-13 Greenberger, Martin
A New Methodology for Computer Simulation
October 1964
AD 609-288

*TR-14 Roos, Daniel
Use of CTSS in a Teaching Environment
November 1964
AD 661-807

*TR-16 Saltzer, Jerome H.
CTSS Technical Notes
March 1965
AD 612-702

*TR-17 Samuel, Arthur L.
Time-Sharing on a Multiconsole Computer
March 1965
AD 462-158

*TR-18 Scherr, Allan Lee
An Analysis of Time-Shared Computer Systems,
Ph.D. Thesis, EE Dept.
June 1965
AD 470-715

*TR-19 Russo, Francis John
A Heuristic Approach to Alternate Routing in a Job Shop,
B.S. & M.S. Theses, Sloan School
June 1965
AD 474-018

*TR-20 Wantman, Mayer Elihu
CALCULAID: An On-Line System for Algebraic Computation and Analysis,
M.S. Thesis, Sloan School
September 1965
AD 474-019

*TR-21 Denning, Peter James
Queueing Models for File Memory Operation,
M.S. Thesis, EE Dept.
October 1965

AD 624-943

*TR-22 Greenberger, Martin
The Priority Problem
November 1965

AD 625-728

*TR-23 Dennis, Jack B., and Earl C. Van Horn
Programming Semantics for Multi-programmed Computations
December 1965

AD 627-537

*TR-24 Kaplow, Roy, Stephen Strong and John Brackett
MAP: A System for On-Line Mathematical Analysis
January 1966

AD 476-443

*TR-25 Stratton, William David
Investigation of an Analog Technique to Decrease Pen-Tracking Time in Computer Displays,
M.S. Thesis, EE Dept.
March 1966

AD 631-396

*TR-26 Cheek, Thomas Burrell
Design of a Low-Cost Character Generator for Remote Computer Displays,
M.S. Thesis, EE Dept.
March 1966

AD 631-269

*TR-27 Edwards, Daniel James
OCAS - On-Line Cryptanalytic Aid System,
M.S. Thesis, EE Dept.
May 1966

AD 633-678

*TR-28 Smith, Arthur Anshel
Input/Output in Time-Shared, Segmented, Multiprocessor Systems,
M.S. Thesis, EE Dept.
June 1966
AD 637-215

*TR-29 Ivie, Evan Leon
Search Procedures Based on Measures of Relatedness between Documents,
Ph.D. Thesis, EE Dept.
June 1966
AD 636-275

*TR-30 Saltzer, Jerome Howard
Traffic Control in a Multiplexed Computer System,
Sc.D. Thesis, EE Dept.
July 1966
AD 635-966

*TR-31 Smith, Donald L.
Models and Data Structures for Digital Logic Simulation,
M.S. Thesis, EE Dept.
August 1966
AD 637-192

*TR-32 Teitelman, Warren
PILOT: A Step Toward Man-Computer Symbiosis,
Ph.D. Thesis, Math. Dept.
September 1966
AD 638-446

*TR-33 Norton, Lewis M.
ADEPT - A Heuristic Program for Proving Theorems of Group Theory,
Ph.D. Thesis, Math. Dept.
October 1966
AD 645-660

*TR-34 Van Horn, Earl C., Jr.
Computer Design for Asynchronously Reproducible Multiprocessing,
Ph.D. Thesis, EE Dept.
November 1966
AD 650-407

*TR-35 Fenichel, Robert R.
An On-Line System for Algebraic Manipulation,
Ph.D. Thesis, Appl. Math. (Harvard)
December 1966

AD 657-282

*TR-36 Martin, William A.
Symbolic Mathematical Laboratory,
Ph.D. Thesis, EE Dept.
January 1967

AD 657-283

*TR-37 Guzman-Arenas, Adolfo
Some Aspects of Pattern Recognition
by Computer,
M.S. Thesis, EE Dept.
February 1967

AD 656-041

*TR-38 Rosenberg, Ronald C., Daniel W. Kennedy
and Roger A. Humphrey
A Low-Cost Output Terminal For Time-
Shared Computers
March 1967

AD 662-027

*TR-39 Forte, Allen
Syntax-Based Analytic Reading of
Musical Scores
April 1967

AD 661-806

*TR-40 Miller, James R.
On-Line Analysis for Social Scientists
May 1967

AD 668-009

*TR-41 Coons, Steven A.
Surfaces for Computer-Aided Design
of Space Forms
June 1967

AD 663-504

*TR-42 Liu, Chung L., Gabriel D. Chang
and Richard E. Marks
Design and Implementation of a Table-
Driven Compiler System
July 1967

AD 668-960

*TR-43 Wilde, Daniel U.
Program Analysis by Digital Computer,
Ph.D. Thesis, EE Dept.
August 1967
AD 662-224

*TR-44 Gorry, G. Anthony
A System for Computer-Aided Diagnosis,
Ph.D. Thesis, Sloan School
September 1967
AD 662-665

*TR-45 Leal-Cantu, Nestor
On the Simulation of Dynamic Systems
with Lumped Parameters and Time Delays,
M.S. Thesis, ME Dept.
October 1967
AD 663-502

*TR-46 Alsop, Joseph W.
A Canonic Translator,
B.S. Thesis, EE Dept.
November 1967
AD 663-503

*TR-47 Moses, Joel
Symbolic Integration,
Ph.D. Thesis, Math. Dept.
December 1967
AD 662-666

*TR-48 Jones, Malcolm M.
Incremental Simulation on a Time-
Shared Computer,
Ph.D. Thesis, Sloan School
January 1968
AD 662-225

*TR-49 Luconi, Fred L.
Asynchronous Computational Structures,
Ph.D Thesis, EE Dept.
February 1968
AD 667-602

*TR-50 Denning, Peter J.
Resource Allocation in Multiprocess
Computer Systems,
Ph.D. Thesis, EE Dept.
May 1968
AD 675-554

*TR-51 Charniak, Eugene
CARPS, A Program which Solves Calculus Word Problems,
M.S. Thesis, EE Dept.
July 1968
AD 673-670

*TR-52 Deitel, Harvey M.
Absentee Computations in a Multiple-Access Computer System,
M.S. Thesis, EE Dept.
August 1968
AD 684-738

*TR-53 Slutz, Donald R.
The Flow Graph Schemata Model of Parallel Computation,
Ph.D. Thesis, EE Dept.
September 1968
AD 683-393

*TR-54 Grochow, Jerrold M.
The Graphic Display as an Aid in the Monitoring of a Time-Shared Computer System,
M.S. Thesis, EE Dept.
October 1968
AD 689-468

*TR-55 Rappaport, Robert L.
Implementing Multi-Process Primitives in a Multiplexed Computer System,
M.S. Thesis, EE Dept.
November 1968
AD 689-469

*TR-56 Thornhill, Daniel E., Robert H. Stotz, Douglas T. Ross and John E. Ward (ESL-R-356)
An Integrated Hardware-Software System for Computer Graphics in Time-Sharing
December 1968
AD 685-202

*TR-57 Morris, James H.
Lambda-Calculus Models of Programming Languages,
Ph.D. Thesis, Sloan School
December 1968
AD 683-394

*TR-58 Greenbaum, Howard J.
A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System,
M.S. Thesis, EE Dept.
January 1969

AD 686-988

*TR-59 Guzman, Adolfo
Computer Recognition of Three-Dimensional Objects in a Visual Scene,
Ph.D. Thesis, EE Dept.
December 1968

AD 692-200

*TR-60 Ledgard, Henry F.
A Formal System for Defining the Syntax and Semantics of Computer Languages,
Ph.D. Thesis, EE Dept.
April 1969

AD 689-305

*TR-61 Baecker, Ronald M.
Interactive Computer-Mediated Animation,
Ph.D. Thesis, EE Dept.
June 1969

AD 690-887

*TR-62 Tillman, Coyt C., Jr. (ESL-R-395)
EPS: An Interactive System for Solving Elliptic Boundary-Value Problems with Facilities for Data Manipulation and General-Purpose Computation
June 1969

AD 692-462

*TR-63 Brackett, John W., Michael Hammer and Daniel E. Thornhill
Case Study in Interactive Graphics Programming: A Circuit Drawing and Editing Program for Use with a Storage-Tube Display Terminal
October 1969

AD 699-930

*TR-64 Rodriguez, Jorge E. (ESL-R-398)
A Graph Model for Parallel Computations,
Sc.D. Thesis, EE Dept.
September 1969

AD 697-759

*TR-65 DeRemer, Franklin L.
Practical Translators for LR(k)
Languages,
Ph.D. Thesis, EE Dept.
October 1969

AD 699-501

*TR-66 Beyer, Wendell T.
Recognition of Topological Invariants
by Iterative Arrays,
Ph.D. Thesis, Math. Dept.
October 1969

AD 699-502

*TR-67 Vanderbilt, Dean H.
Controlled Information Sharing in
a Computer Utility,
Ph.D. Thesis, EE Dept.
October 1969

AD 699-503

*TR-68 Selwyn, Lee L.
Economies of Scale in Computer Use:
Initial Tests and Implications for
The Computer Utility,
Ph.D. Thesis, Sloan School
June 1970

AD 710-011

*TR-69 Gertz, Jeffrey L.
Hierarchical Associative Memories
for Parallel Computation,
Ph.D. Thesis, EE Dept.
June 1970

AD 711-091

*TR-70 Fillat, Andrew I., and Leslie A. Kraning
Generalized Organization of Large
Data-Bases: A Set-Theoretic
Approach to Relations,
B.S. & M.S. Theses, EE Dept.
June 1970

AD 711-060

*TR-71 Fiasconaro, James G.
A Computer-Controlled Graphical Display Processor,
M.S. Thesis, EE Dept.
June 1970
AD 710-479

TR-72 Patil, Suhas S.
Coordination of Asynchronous Events,
Sc.D. Thesis, EE Dept.
June 1970
AD 711-763

*TR-73 Griffith, Arnold K.
Computer Recognition of Prismatic Solids,
Ph.D. Thesis, Math. Dept.
August 1970
AD 712-069

TR-74 Edelberg, Murray
Integral Convex Polyhedra and an Approach to Integralization,
Ph.D. Thesis, EE Dept.
August 1970
AD 712-070

*TR-75 Hebalkar, Prakash G.
Deadlock-Free Sharing of Resources in Asynchronous Systems,
Sc.D. Thesis, EE Dept.
September 1970
AD 713-139

*TR-76 Winston, Patrick H.
Learning Structural Descriptions from Examples,
Ph.D. Thesis, EE Dept.
September 1970
AD 713-988

TR-77 Haggerty, Joseph P.
Complexity Measures for Language Recognition by Canonic Systems,
M.S. Thesis, EE Dept.
October 1970
AD 715-134

*TR-78 Madnick, Stuart E.
Design Strategies for File Systems,
M.S. Thesis, EE Dept. & Sloan School
October 1970
AD 714-269

TR-79 Horn, Berthold K.
Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View,
Ph.D. Thesis, EE Dept.
November 1970
AD 717-336

TR-80 Clark, David D., Robert M. Graham, Jerome H. Saltzer and Michael D. Schroeder
The Classroom Information and Computing Service
January 1971
AD 717-857

*TR-81 Banks, Edwin R.
Information Processing and Transmission in Cellular Automata,
Ph.D. Thesis, ME Dept.
January 1971
AD 717-951

*TR-82 Krakauer, Lawrence J.
Computer Analysis of Visual Properties of Curved Objects,
Ph.D. Thesis, EE Dept.
May 1971
AD 723-647

*TR-83 Lewin, Donald E.
In-Process Manufacturing Quality Control,
Ph.D. Thesis, Sloan School
January 1971
AD 720-098

*TR-84 Winograd, Terry
Procedures as a Representation for Data in a Computer Program for Understanding Natural Language,
Ph.D. Thesis, Math. Dept.
February 1971
AD 721-399

*TR-85 Miller, Perry L.
Automatic Creation of a Code Generator from a Machine Description,
E.E. Thesis, EE Dept.
May 1971
AD 724-730

*TR-86 Schell, Roger R.
Dynamic Reconfiguration in a Modular Computer System,
Ph.D. Thesis, EE Dept.
June 1971
AD 725-859

TR-87 Thomas, Robert H.
A Model for Process Representation and Synthesis,
Ph.D. Thesis, EE Dept.
June 1971
AD 726-049

TR-88 Welch, Terry A.
Bounds on Information Retrieval Efficiency in Static File Structures,
Ph.D. Thesis, EE Dept.
June 1971
AD 725-429

TR-89 Owens, Richard C., Jr.
Primary Access Control in Large-Scale Time-Shared Decision Systems,
M.S. Thesis, Sloan School
July 1971
AD 728-036

TR-90 Lester, Bruce P.
Cost Analysis of Debugging Systems,
B.S. & M.S. Theses, EE Dept.
September 1971
AD 730-521

*TR-91 Smoliar, Stephen W.
A Parallel Processing Model of Musical Structures,
Ph.D. Thesis, Math. Dept.
September 1971
AD 731-690

TR-92 Wang, Paul S.
Evaluation of Definite Integrals
by Symbolic Manipulation
Ph.D. Thesis, Math. Dept.
October 1971

AD 732-005

TR-93 Greif, Irene Gloria
Induction in Proofs about Programs,
M.S. Thesis, EE Dept.
February 1972.

AD 737-701

TR-94 Hack, Michel Henri Theodore
Analysis of Production Schemata
by Petri Nets,
M.S. Thesis, EE Dept.
February 1972

AD 740-320

*TR-95 Fateman, Richard J.
Essays in Algebraic Simplification
(A revision of a Harvard Ph.D. Thesis)
April 1972

AD 740-132

TR-96 Manning, Frank
Autonomous, Synchronous Counters Constructed Only of
J-K Flip-Flops,
M.S. Thesis, EE Dept.
May 1972

AD 744-030

TR-97 Vilfan, Bostjan
The Complexity of Finite Functions
Ph.D. Thesis, EE Dept.
March 1972

AD 739-678

TR-98 Stockmeyer, Larry Joseph
Bounds on Polynomial Evaluation Algorithms
M.S. Thesis, EE Dept.
April 1972

AD 740-328

TR-99 Lynch, Nancy Ann
Relativization of the Theory of Computational Complexity
Ph.D. Thesis, Math. Dept.
June 1972
AD 744-032

TR-100 Mandl, Robert
Further Results on Hierarchies of Canonic Systems
M.S. Thesis, EE Dept.
June 1972
AD 744-206

TR-101 Dennis, Jack B.
On the Design and Specification of a Common Base Language
June 1972
AD 744-207

TR-102 Hossley, Robert F.
Finite Tree Automata and ω-Automata
M.S. Thesis, EE Dept.
September 1972
AD 749-367

*TR-103 Sekino, Akira
Performance Evaluation of Multiprogrammed Time-Shared Computer Systems
Ph.D Thesis, EE Dept.
September 1972
AD 749-949

TR-104 Schroeder, Michael D.
Cooperation of Mutually Suspicious Subsystems in a Computer Utility
Ph.D. Thesis, EE Dept.
September 1972
AD 750-173

TR-105 Smith, Burton J.
An Analysis of Sorting Networks
Sc.D. Thesis, EE Dept.
October 1972
AD 751-614

TR-106 Rackoff, Charles W.
The Emptiness and Complementation Problems for Automata on Infinite Trees
M.S. Thesis, EE Dept.
January 1973
AD 756-248

TR-107 Madnick, Stuart E.
Storage Hierarchy Systems
Ph.D. Thesis, EE Dept.
April 1973
AD 760-001

TR-108 Wand, Mitchell
Mathematical Foundations of Formal Language Theory
Ph.D. Thesis, Math. Dept.
December 1973

TR-109 Johnson, David S.
Near-Optimal Bin Packing Algorithms
Ph.D. Thesis, Math. Dept.
June 1973
PB 222-090

TR-110 Moll, Robert
Complexity Classes of Recursive Functions
Ph.D. Thesis, Math. Dept.
June 1973
AD 767-730

TR-111 Linderman, John P.
Productivity in Parallel Computation Schemata
Ph.D. Thesis, EE Dept.
December 1973
PB 226-159/AS

TR-112 Hawryszkiewycz, Igor T.
Semantics of Data Base Systems
Ph.D. Thesis, EE Dept.
December 1973
PB 226-061/AS

TR-113 Herrmann, Paul P.
On Reducibility Among Combinatorial Problems
M.S. Thesis, Math. Dept.
December 1973
PB 226-157/AS

TR-114 Metcalfe, Robert M.
Packet Communication
Ph.D. Thesis, Applied Math., Harvard University
December 1973
AD 771-430

TR-115 Rotenberg, Leo
Making Computers Keep Secrets
Ph.D Thesis, EE Dept.
February 1974
PB 229-352/AS

TR-116 Stern, Jerry A.
Backup and Recovery of On-Line Information in a Computer Utility
M.S. & E.E. Theses, EE Dept.
January 1974
AD 774-141

TR-117 Clark, David D.
An Input/Output Architecture for Virtual Memory Computer Systems
Ph.D. Thesis, EE Dept.
January 1974
AD 774-738

TR-118 Briabrin, Victor
An Abstract Model of a Research Institute: Simple Automatic Programming Approach
March 1974
PB 231-505/AS

TR-119 Hammer, Michael M.
A New Grammatical Transformation into Deterministic Top-Down Form
Ph.D. Thesis, EE Dept.
February 1974
AD 775-545

TR-120 Ramchandani, Chander
Analysis of Asynchronous Concurrent Systems by Timed Petri Nets
Ph.D. Thesis, EE Dept.
February 1974
AD 775-618

TR-121 Yao, Foong F.
On Lower Bounds for Selection Problems
Ph.D. Thesis, Math. Dept.
March 1974
PB 230-950/AS

TR-122 Scherf, John A.
Computer and Data Security: A Comprehensive Annotated Bibliography
M.S. Thesis, Sloan School
January 1974
AD 775-546

TR-123 Introduction to Multics
February 1974
AD 918-562

TR-124 Laventhal, Mark S.
Verification of Programs Operating on Structured Data
B.S. & M.S. Theses, EE Dept.
March 1974
PB 231-365/AS

TR-125 Mark, William S.
A Model-Debugging System
B.S. & M.S. Theses, EE Dept.
April 1974
AD 778-688

TR-126 Altman, Vernon E.
A Language Implementation System
B.S. & M.S. Theses, Sloan School
May 1974
AD 780-672

TR-127 Greenberg, Bernard S.
An Experimental Analysis of Program Reference Patterns in the Multics Virtual Memory
M.S. Thesis, EE Dept.
May 1974
AD 780-407

TR-128 Frankston, Robert M.
The Computer Utility as a Marketplace for Computer Services
M.S. & E.E. Theses, EE Dept.
May 1974
AD 780-436

TR-129 Weissberg, Richard W.
Using Interactive Graphics in Simulating the Hospital Emergency Room
M.S. Thesis, EE Dept.
May 1974
AD 780-437

TR-130 Ruth, Gregory R.
Analysis of Algorithm Implementations
Ph.D. Thesis, EE Dept.
May 1974
AD 780-408

TR-131 Levin, Michael
Mathematical Logic for Computer Scientists
June 1974

TR-132 Janson, Philippe A.
Removing the Dynamic Linker from the Security Kernel of a Computing Utility
M.S. Thesis, EE Dept.
June 1974
AD 781-305

TR-133 Stockmeyer, Larry J.
The Complexity of Decision Problems in Automata Theory and Logic
Ph.D. Thesis, EE Dept.
July 1974
PB 235-283/AS

*TR-134 Ellis, David J.
Semantics of Data Structures and References
M.S. & E.E. Theses, EE Dept.
August 1974
PB 236-594/AS

TR-135 Pfister, Gregory F.
The Computer Control of Changing Pictures
Ph.D. Thesis, EE Dept.
September 1974
AD 787-795

TR-136 Ward, Stephen A.
Functional Domains of Applicative Languages
Ph.D. Thesis, EE Dept.
September 1974
AD 787-796

TR-137 Seiferas, Joel I.
Nondeterministic Time and Space Complexity Classes
Ph.D Thesis, Math. Dept.
September 1974
PB 236-777/AS

TR-138 Yun, David Y. Y.
The Hensel Lemma in Algebraic Manipulation
Ph.D. Thesis, Math. Dept.
November 1974
AD A002-737

TR-139 Ferrante, Jeanne
Some Upper and Lower Bounds on Decision Procedures in Logic
Ph.D. Thesis, Math. Dept.
November 1974
PB 238-121/AS

TR-140 Redell, David D.
Naming and Protection in Extendible Operating Systems
Ph.D. Thesis, EE Dept.
November 1974
AD A001-721

TR-141 Richards, Martin, A. Evans and R. Mabee
The BCPL Reference Manual
December 1974
AD A003-599

TR-142 Brown, Gretchen P.
Some Problems in German to English Machine Translation
M.S. & E.E. Theses, EE Dept.
December 1974
AD A003-002

TR-143 Silverman, Howard
A Digitalis Therapy Advisor
M.S. Thesis, EE Dept.
January 1975

TR-144 Rackoff, Charles
The Computational Complexity of Some Logical Theories
Ph.D. Thesis, EE Dept.
February 1975

*TR-145 Henderson, D. Austin
The Binding Model: A Semantic Base for Modular Programming Systems
Ph.D. Thesis, EE Dept.
February 1975
AD A006-961

*TR-146 Malhotra, Ashok
Design Criteria for a Knowledge-Based
English Language System for Management:
An Experimental Analysis
Ph.D. Thesis, EE Dept.
February 1975

TR-147 Van De Vanter, Michael L.
A Formalization and Correctness Proof
of the CGOL Language System
M.S. Thesis, EE Dept.
March 1975

TR-148 Johnson, Jerry
Program Restructuring for Virtual Memory Systems
Ph.D. Thesis, EE Dept.
March 1975
AD A009-218

*TR-149 Snyder, Alan
A Portable Compiler for the Language C
B.S. & M.S. Theses, EE Dept.
May 1975
AD A010-218

*TR-150 Rumbaugh, James E.
A Parallel Asynchronous Computer Architecture
for Data Flow Programs
Ph.D. Thesis, EE Dept.
May 1975
AD A010-918

TR-151 Manning, Frank B.
Automatic Test, Configuration, and Repair
of Cellular Arrays
Ph.D. Thesis, EE Dept.
June 1975
AD A012-822

TR-152 Qualitz, Joseph E.
Equivalence Problems for Monadic Schemas
Ph.D. Thesis, EE Dept.
June 1975
AD A012-823

TR-153 Miller, Peter B.
Strategy Selection in Medical Diagnosis
M.S. Thesis, EE & CS Dept.
September 1975

TR-154 Greif, Irene
Semantics of Communicating Parallel Processes
Ph.D. Thesis, EE & CS Dept.
September 1975
AD A016-302

TR-155 Kahn, Kenneth M.
Mechanization of Temporal Knowledge
M.S. Thesis, EE & CS Dept.
September 1975

TR-156 Bratt, Richard G.
Minimizing the Naming Facilities Requiring Protection in a Computer Utility
M.S. Thesis, EE & CS Dept.
September 1975

*TR-157 Meldman, Jeffrey A.
A Preliminary Study in Computer-Aided Legal Analysis
Ph.D. Thesis, EE & CS Dept.
November 1975
AD A018-997

TR-158 Grossman, Richard W.
Some Data-base Applications of Constraint Expressions
M.S. Thesis, EE & CS Dept.
February 1976
AD A024-149

TR-159 Hack, Michel
Petri Net Languages
March 1976

TR-160 Bosyj, Michael
A Program for the Design of Procurement Systems
M.S. Thesis, EE & CS Dept.
May 1976
AD A026-688

TR-161 Hack, Michel
Decidability Questions
Ph.D. Thesis, EE & CS Dept.
June 1976

*TR-162 Kent, Stephen T.
Encryption-Based Protection Protocols for Interactive User-Computer Communication
M.S. Thesis, EE & CS Dept.
June 1976
AD A026-911

TR-163 Montgomery, Warren A.
A Secure and Flexible Model of Process Initiation for a Computer Utility
M.S. & E.E. Theses, EE & CS Dept.
June 1976

TR-164 Reed, David P.
Processor Multiplexing in a Layered Operating System
M.S. Thesis, EE & CS Dept.
July 1976

TR-165 McLeod, Dennis J.
High Level Expression of Semantic Integrity Specifications in a Relational Data Base System
M.S. Thesis, EE & CS Dept.
September 1976
AD A034-184

TR-166 Chan, Arvola Y.
Index Selection in a Self-Adaptive Relational Data Base Management System
M.S. Thesis, EE & CS Dept.
September 1976
AD A034-185

TR-167 Janson, Philippe A.
Using Type Extension to Organize Virtual Memory Mechanisms
Ph.D. Thesis, EE & CS Dept.
September 1976

TR-168 Pratt, Vaughan R.
Semantical Considerations on Floyd-Hoare Logic
September 1976

TR-169 Safran, Charles, James F. Desforges and Philip N. Tsichlis
Diagnostic Planning and Cancer Management
September 1976

TR-170 Furtek, Frederick C.
The Logic of Systems
Ph.D. Thesis, EE & CS Dept.
December 1976

TR-171 Huber, Andrew R.
A Multi-Process Design of a Paging System
M.S. & E.E. Theses, EE & CS Dept.
December 1976

TR-172 Mark, William S.
The Reformulation Model of Expertise
Ph.D. Thesis, EE & CS Dept.
December 1976

AD A035-397

TR-173 Goodman, Nathan
Coordination of Parallel Processes in the Actor Model of Computation
M.S. Thesis, EE & CS Dept.
December 1976

TR-174 Hunt, Douglas H.
A Case Study of Intermodule Dependencies in a Virtual Memory Subsystem
M.S. & E.E. Theses, EE & CS Dept.
December 1976

TR-175 Goldberg, Harold J.
A Robust Environment for Program Development
M.S. Thesis, EE & CS Dept.
February 1977

TR-176 Swartout, William R.
A Digitalis Therapy Advisor with Explanations
M.S. Thesis, EE & CS Dept.
February 1977

TR-177 Mason, Andrew H.
A Layered Virtual Memory Manager
M.S. & E.E. Theses, EE & CS Dept.
May 1977

*TR-178 Bishop, Peter B.
Computer Systems with a Very Large Address Space and Garbage Collection
Ph.D. Thesis, EE & CS Dept.
May 1977

AD A040-601

TR-179 Karger, Paul A.
Non-Discretionary Access Control for Decentralized Computing Systems
M.S. Thesis, EE & CS Dept.
May 1977
AD A040-804

TR-180 Luniewski, Allen W.
A Simple and Flexible System Initialization Mechanism
M.S. & E.E. Theses, EE & CS Dept.
May 1977

TR-181 Mayr, Ernst W.
The Complexity of the Finite Containment Problem for Petri Nets
M.S. Thesis, EE & CS Dept.
June 1977

TR-182 Brown, Gretchen P.
A Framework for Processing Dialogue
June 1977
AD A042-370

TR-183 Jaffe, Jeffrey M.
Semilinear Sets and Applications
M.S. Thesis, EE & CS Dept.
July 1977

*TR-184 Levine, Paul H.
Facilitating Interprocess Communication in a Heterogeneous Network Environment
B.S. & M.S. Theses, EE & CS Dept.
July 1977
AD A043-901

TR-185 Goldman, Barry
Deadlock Detection in Computer Networks
B.S. & M.S. Theses, EE & CS Dept.
September 1977
AD A047-025

TR-186 Ackerman, William B.
A Structure Memory for Data Flow Computers
M.S. Thesis, EE & CS Dept.
September 1977
AD A047-026

TR-187 Long, William J.
A Program Writer
Ph.D. Thesis, EE & CS Dept.
November 1977
AD A047-595

TR-188 Bryant, Randal E.
Simulation of Packet Communication Architecture Computer Systems
M.S. Thesis, EE & CS Dept.
November 1977
AD A048-290

TR-189 Ellis, David J.
Formal Specifications for Packet Communication Systems
Ph.D. Thesis, EE & CS Dept.
November 1977
AD A048-980

TR-190 Moss, J. Eliot B.
Abstract Data Types in Stack Based Languages
M.S. Thesis, EE & CS Dept.
February 1978
AD A052-332

TR-191 Yonezawa, Akinori
Specification and Verification Techniques for Parallel Programs Based on Message Passing Semantics
Ph.D. Thesis, EE & CS Dept.
January 1978
AD A051-149

TR-192 Niamir, Bahram
Attribute Partitioning in a Self-Adaptive Relational Database System
M.S. Thesis, EE & CS Dept.
January 1978
AD A053-292

TR-193 Schaffert, J. Craig
A Formal Definition of CLU
M.S. Thesis, EE & CS Dept.
January 1978

TR-194 Hewitt, Carl and Henry Baker, Jr.
Actors and Continuous Functionals
February 1978
AD A052-266

TR-195 Bruss, Anna R.
On Time-Space Classes and Their Relation to the Theory of Real Addition
M.S. Thesis, EE & CS Dept.
March 1978

TR-196 Schroeder, Michael D., David D. Clark, Jerome H. Saltzer and Douglas H. Wells
Final Report of the Multics Kernel Design Project
March 1978

TR-197 Baker, Henry Jr.
Actor Systems for Real-Time Computation
Ph.D. Thesis, EE & CS Dept.
March 1978
AD A053-328

TR-198 Halstead, Robert H., Jr.
Multiple-Processor Implementation of Message-Passing Systems
M.S. Thesis, EE & CS Dept.
April 1978
AD A054-009

TR-199 Terman, Christopher J.
The Specification of Code Generation Algorithms
M.S. Thesis, EE & CS Dept.
April 1978
AD A054-301

TR-200 Harel, David
Logics of Programs: Axiomatics and Descriptive Power
Ph.D. Thesis, EE & CS Dept.
May 1978

TR-201 Scheifler, Robert W.
A Denotational Semantics of CLU
M.S. Thesis, EE & CS Dept.
June 1978

PROGRESS REPORTS

*Project MAC Progress Report I
to July 1964

AD 465-088

*Project MAC Progress Report II
July 1964-July 1965

AD 629-494

*Project MAC Progress Report III
July 1965-July 1966

AD 648-346

*Project MAC Progress Report IV
July 1966-July 1967

AD 681-342

*Project MAC Progress Report V
July 1967-July 1968

AD 687-770

*Project MAC Progress Report VI
July 1968-July 1969

AD 705-434

*Project MAC Progress Report VII
July 1969-July 1970

AD 732-767

*Project MAC Progress Report VIII
July 1970-July 1971

AD 735-148

*Project MAC Progress Report IX
July 1971-July 1972

AD 756-689

*Project MAC Progress Report X
July 1972-July 1973

AD 771-428

*Project MAC Progress Report XI
July 1973-July 1974

AD A004-966

*Laboratory for Computer Science Progress Report XII
July 1974-July 1975

AD A024-527

*Laboratory for Computer Science Progress Report XIII
July 1975-July 1976

AD A061-246

Laboratory for Computer Science Progress Report XIV
July 1976-July 1977

AD A061-932

Copies of all reports with AD and PB numbers listed in Publications may be secured from the National Technical Information Service, Operations Division, Springfield, Virginia, 22151. Prices vary. The AD or PB number must be supplied with the request.

* Out of Print reports may be obtained from NTIS if the AD number is supplied (see above). Out of Print reports without an AD or PB number are unobtainable.

OFFICIAL DISTRIBUTION LIST

Defense Documentation Center
Cameron Station
Alexandria, VA 22314
12 copies

Office of Naval Research
Information Systems Program
Code 437
Arlington, VA 22217
2 copies

Office of Naval Research
Branch Office/Boston
Building 114, Section D
666 Summer Street
Boston, MA 02210
1 copy

Office of Naval Research
Branch Office/Chicago
536 South Clark Street
Chicago, IL 60605
1 copy

Office of Naval Research
Branch Office/Pasadena
1030 East Green Street
Pasadena, CA 91106
1 copy

New York Area
715 Broadway - 5th floor
New York, N. Y. 10003
1 copy

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D. C. 20375
6 copies

Assistant Chief for Technology
Office of Naval Research
Code 200
Arlington, VA 22217
1 copy

Office of Naval Research
Code 455
Arlington, VA 22217
1 copy

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, D. C. 20380
1 copy

Office of Naval Research
Code 458
Arlington, VA 22217
1 copy

Naval Ocean Systems Center
Advanced Software Techonolgy
Division - Code 5200
San Diego, CA 92152
1 copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation & Math Department
Bethesda, MD 20084
1 copy

Captain Grace M. Hopper (008)
Naval Data Automation Command
Washington Navy Yard
Building 166
Washington, D. C. 20374
1 copy

Mr. Kin B. Thompson
Technical Director
Information Systems Division
(OP-91T)
Office of Chief of Naval Operations
Washington, D. C. 20350
1 copy

Captain Richard L. Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York, N. Y. 09501
1 copy